THE DESIGN OF A MICROCOMPUTER BASED GENERALIZED PROCESS CONTROL SYSTEM

by

Frank Joseph Nelson

A thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science of Electrical Engineering

University of Washington

1982

MASTER'S THESIS

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this thesis is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U. S. Copyright law. Any other reproduction for any purposes or by any means shall not be allowed without my written permission.

University of Washington

Abstract

THE DESIGN OF A MICROCOMPUTER BASED GENERALIZED PROCESS CONTROL SYSTEM

By Frank Joseph Nelson

Chairperson of the Supervisory Committee:

Professor William E. Moritz
Department of Electrical Engineering

Physical processes, involving the movement of fluids or materials in time and space, require feedback control systems to maintain process variables such as flowrate, temperature and pressure at predetermined values to enable the products of the process to be correct. For many years this area of control was dominated by analog control devices, both electronic and mechanical. With the advent of minicomputers in the 1960's large plants converted from using many individual controllers to using a central computer to control all process variables. The introduction of microprocessors in the 1970's has brought a shift in smaller plants from individual analog controllers to individual and multivariable microprocessor control systems.

This design provides a very generalized controller, using an Intel single board microcomputer, a 16 channel analog-to-digital converter, a 4 channel digital-to-analog converter, a floating point arithmetic processor and a 16K-byte memory expansion board to implement a process control system. The software is resident in 8K-bytes of Read Only Memory and provides 4 Modes of Operation.

The Modes of Operation available to the user are: 1) File Mode, which offers an easy to use, menu-oriented, interactive way to define a desired process control strategy; 2) Modify Mode, which enables simple and quick interactive editing and revision of a process control strategy; 3) Run Mode, which translates the control strategy into a real time process control program and executes it; and 4) Terminal Mode, in which the system becomes a remote terminal of a larger computer.

The key feature of this system is the fact that the user does not write a computer program to obtain real time computer control of the process. Instead, the user provides the process control system with elements of the mathematical description, or configuration, of the control system desired, and the process control system applies these elements to a generalized real time control program resident in the system. This program, now tailored for a specific process, is used to actually sample inputs from

the process and provide the needed signals to control the process.

TABLE OF CONTENTS

LIST OF FIGURES iii

LIST OF TABLES v

Chapter 1. PROJECT DEFINITION 1

INTRODUCTION 1

PROBLEM STATEMENT 7

Chapter 2. DESIGN SPECIFICATIONS 12

OVERVIEW 12

SYSTEM SPECIFICATIONS 14

Chapter 3. HARDWARE DESIGN 19

HARDWARE REQUIREMENTS 19

HARDWARE SELECTION 23

HARDWARE DESCRIPTION 25

HARDWARE CONNECTION DATA 37

Chapter 4. SOFTWARE DESIGN 44

INTRODUCTION 44

SOFTWARE ORGANIZATION 44

SOFTWARE DESCRIPTION 52

PROGRAM LISTING 71

Chapter 5. CONCLUSIONS 72

SYSTEM TESTING 72

ADVANTAGES OF THE SYSTEM 73

DISADVANTAGES OF THE SYSTEM 75

POTENTIAL IMPROVEMENTS 76

LIST OF REFERENCES 78

Appendix A. USER'S MANUAL 80

Appendix B. MEMORY MAP 88

LIST OF FIGURES

Number		Page
1.	Control of a Physical Process	1
2.	Analog Temperature Controller	4
3.	Block Representation of Process Control	9
4.	Process Control System, Simplified Block Diagram	19
5.	Process Control System, Detailed Block Diagram	26
6.	SBC 80/24 Single Board Computer, Block Diagram	28
7.	SBC 116A Expansion Board, Block Diagram	30
8.	SBC 711 Analog Input Board, Block Diagram	33
9.	SBC 724 Analog Output Board, Block Diagram	35
10.	System Connection Panel	43
11.	Software Organization, System Routines	46
12.	Software Organization, Configuration Functions	49
13.	Software Organization, Run Functions	51
14.	System Menu	55
15.	Configuration File Routine, Defining a Block	60
16.	Configuration File Routine, Defining an Output	61
17.	Run Configuration Routine, Real Time Display	65
A. 1.	System Block Diagram	81
A. 2.	Single Loop Feedback Example	81
A. 3.	System Menu	84
A. 4.	Example of Block Definition	86

A.5. Example of Run Mode Display 87

LIST OF TABLES

Number		Page
1.	Hardware Configuration Alternatives	24
2.	Input/Output Port Assignments	38
3.	Process Control System, Address Assignments	39
4.	Process Control System, Input Connections	41
5.	Process Control System, Output Connections	42
6.	Configuration File Structure	47
7.	Run Time Tables	48
B. 1.	Process Control System, Memory Map	86

ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to Professor William E. Moritz for his guidance and careful critical review of this effort. I would also like to thank Professor N. L. Ricker of the Chemical Engineering Department for identifying the need for a process control system and for helping to establish the goals of the project. In addition, special thanks go to Douglas K. Medema of the Electrical Engineering Department, without whose assistance in learning to use the logic development systems this project might not have been completed.

I would also like to express appreciation to the Intel Corporation, whose generous donation of equipment made the development of the process control system for the Chemical Engineering Department possible.

For their help and understanding all of the time that I was with this project instead of with them, I am deeply grateful to my wife Lori and my son Eric.

Chapter 1

PROJECT DEFINITION

INTRODUCTION

Process control is the term applied to attaining and maintaining specific conditions in a physical process. These conditions are referred to as process variables, (PV). The desired value of the condition being controlled is called the set point. The five most commonly controlled process variables are flow, level, pressure, temperature and composition. [1] Figure 1 illustrates the relationship between a physical process and the mechanism which is used to control it.

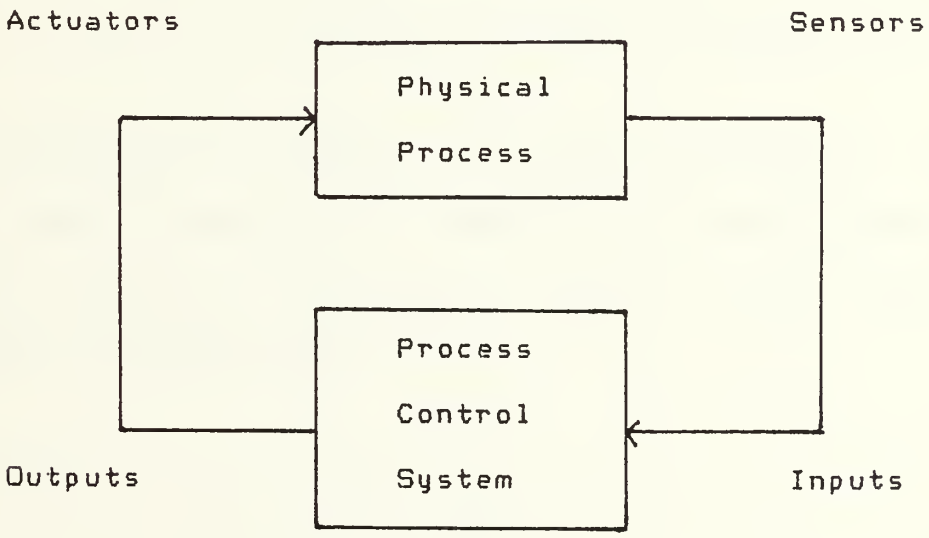


Figure 1. Control of a Physical Process

The mechanism used to control the process variable uses an algorithm to convert the difference between the PV and the set point, the error, into a control signal for the actuator. The most commonly used algorithms include On/Off, Proportional, Proportional plus Integral and Proportional plus Integral plus Derivative. On/Off is simply a switching action. When the PV exceeds the set point, the actuator is turned full on or full off. Proportional control (P) provides a signal that varies directly with the error. Equation (1) describes the P control algorithm.

$$m = K * e \qquad \text{Equation (1)}$$

Where: m is control signal.

K is proportional gain factor.

e is set point - process variable.

Proportional plus integral (PI) control provides a control signal that is the sum of a simple proportional element and the time integral of the error. The PI control algorithm is shown in Equation (2).

$$m = K * \left\{ e + R * \int e\, dt \right\} \qquad \text{Equation (2)}$$

Where: m is control signal.

K is proportional gain factor.

e is set point - process variable.

R is Reset, the reciprocal of the integral time constant.

Proportional plus Integral plus Derivative (PID) control adds to the proportional and integral actions the time rate of change of the error. This derivative action improves dynamic response in many control loops. [2] The PID control algorithm is shown in equation (3).

$$m = K * \left\{ e + R * \int e\, dt + D * de/dt \right\} \qquad \text{Equation (3)}$$

Where: m is control signal.

K is proportional gain factor.

e is set point - process variable.

R is Reset.

D is Rate, the time constant of the derivative action.

For a single variable case, such as temperature, dependent on only a single input, it has been easy to develop simple hardware controls to maintain the process variable at a preset level, such as shown in Figure 2. The potenti-

ometer provides the set point voltage which is compared by the LM 3911 with an internally generated temperature dependent voltage to produce a control signal for the triac in series with the heating element. [3]

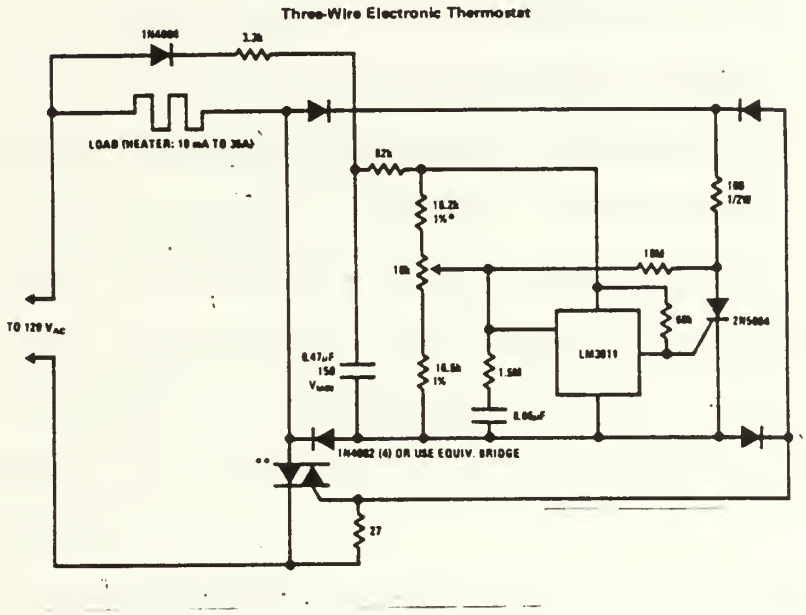


Figure 2. Analog Temperature Controller
(National Semiconductor Linear Databook, p 9-106.)

Process Controllers designed for local control of a single variable become much more complicated with the requirement for user selectable control algorithm, time constants, gain and other parameters. To provide the user with wide ranges in these values, it is difficult and costly to implement controllers in analog hardware. Advances in digital hardware have offered some relief, but the tech-

nique of centralized control of a large number of process variables using a computer system has until recently been the area of primary interest. In the early 1960s large plants began shifting from using a large number of local hardware PID type controllers to using the then new "minicomputers" to sample the process variables for the entire plant and generating appropriate control signals for all of the actuators. A complex system of communications lines, called a data hiway, connected the minicomputer in the plant control room with the various sensors and actuators. [4,5,6] For small plants, the cost of a minicomputer system was prohibitive, so individual analog process controllers continued to fill the majority of process control requirements.

The advent of the microprocessor in the early 1970s, brought the power and versatility of software control down from the large systems to the range of single variable controllers. The microprocessor, a 4, 8, or 16 bit computer central processor, offers extensive software control capability, versatility, small size and low cost. Like the larger computer systems, however, it must be programmed in order to be used. Commercially available units vary greatly in the amount of "programming" involved in making the controller perform the functions needed by the user. The simplest available microprocessor based controllers involve

switch selection of algorithm and parameters from a control panel, in much the same way as similar analog controllers. More versatile units require the user to enter the necessary data in a problem oriented or special application language from a terminal or data entry panel. These units typically provide control for up to 16 variables. Larger microprocessor based controllers frequently use a high level computer language for programming and are competitive with minicomputer systems. [7]

The major difficulty in providing a useable microprocessor based controller, especially one designed to handle a number of process variables and capable of using a variety of control algorithms, is in developing the software for it such that the user does not need to write a computer program in order to use the system. This requires the system designer to develop software which can communicate conversationally with the user to aid the user in selecting the control functions and parameters in much the same manner as he would do if using a classic hardware controller. This research will investigate doing just that, designing a microprocessor based process control system with user-friendly software that will enable the user to easily control a number of process variables using familiar control algorithms without having to wrestle with programming the processor.

PROBLEM STATEMENT

A need exists for such a microcomputer based process control system within the Chemical Engineering Department of the University of Washington. Students in the Process Dynamics and Control classes currently use an analog computer system to simulate the operation of a process and to simulate real time control of the process. However, no actual real time control is available for a physical process. Since a number of types of processes need to be investigated, and for each process several control strategies need to be compared, it would not be feasible to obtain dedicated hardware controllers for every case. A microprocessor based control system offering selectable control algorithms and providing control over several process variables would give the flexibility needed. Students would be using the system to aid them in the design and testing of control strategies, not to learn how to write and debug computer programs for this particular system. As a result, the system must provide the student with a "non-programming" communication format, such as selection from a menu or "fill-in-the-blanks," to use in describing the control strategy desired.

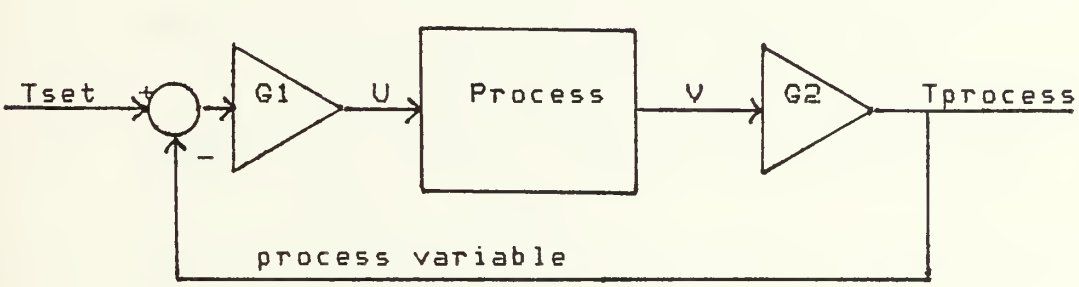
The requirements for a process control system for this generalized application have been developed with the help of Professor N. L. Ricker of the Chemical Engineering De-

partment and can be summarized as follows:

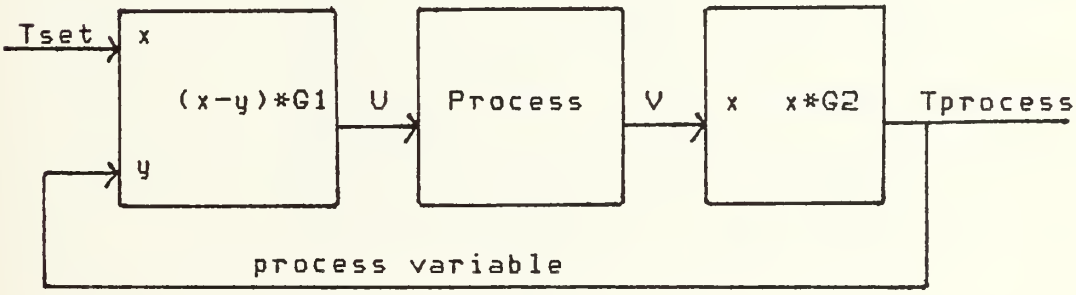
1. The system must be interactive with user via a terminal.
2. The system must provide an easy way for the user to describe the process and the control strategy desired.
3. The user must be able to specify the following:
   a. The sample and control update period.
   b. Which system input channels will be used.
   c. Which system output channels will be used.
   d. What data to collect for later analysis.
   e. The control strategy to use.

The last item, specifying the control strategy, involves considerably more than is immediately apparent. One system of notation used in process control involves functional blocks which represent control actions such as summing two inputs, multiplying two values or providing a control signal proportional to the time integral of the difference between a process variable and a set point. In this notation system, the block is a computational step that follows almost directly from the control strategy. A block consists of its inputs, its output and the operation it performs in between. For example, PI control could be easily defined in terms of one block, a PID block, with a

specified Reset, with Rate equal 0 (to disable derivative action), a specified overall gain, a desired set point, the source of the input variable and the destination of the output control signal. Since potential users are likely to be familiar with this notation, it would be advantageous to them if it were utilized by the system. Figure 3 demonstrates the comparison between the block diagram of a feedback temperature control system and a "block" notation description of the same system.



a. Block Diagram



b. Block Notation

Figure 3. Block Representation of Process Control

The minimum specific block "types" the system should

provide are as follows:

1. ADD Output equals sum of two inputs multiplied by gain factor.
2. SUB Output equals difference of two inputs multiplied by gain factor.
3. MUL Output equals product of two inputs multiplied by gain factor.
4. DIV Output equals quotient of two inputs multiplied by gain factor.
5. SQRT Output equals square root of input multiplied by gain factor. (SQT)
6. SCALE Output equals input multiplied by gain factor. (SCL)
7. PID Proportional-Integral-Derivative Control Block. Output equals sum of three components multiplied by gain factor. User selects 1 of 3 modes, P, PI, PID, by choice of Rate, the derivative time constant, D, and by choice of Reset, the reciprocal integral time constant, R.

The complete description of the resulting process control strategy and the necessary connections to the physical process will be called a "configuration." The system must be able to obtain this information from the user, use it to

tailor a resident real time program to meet the user's needs and, at the direction of the user, execute this program to provide control of the process. To be useful in the expected student environment, the system must provide an easy to use means to revise a configuration once it has been given to the system, either to correct an error or to alter the control strategy. To some extent, this editing feature must also be available during the actual running of the control program. The latter is needed to enable handling of initial conditions, changes in references, and comparison of different selections of gain factors.

One last feature is desirable in the Chemical Engineering instructional environment. The system should be able to operate as a remote terminal of the department's PDP-11/60 computer through a serial RS-232C data link. Using the data link, the students are able to input and execute numerical analysis programs operating on data obtained for them by the process control system. They may also use any other features of the PDP-11 computer, including text editing, file operations and other applications programming.

Chapter 2

DESIGN SPECIFICATIONS

OVERVIEW

To truly be able to handle the "general case" a process control system would need to have an infinite number of communication channels, be able to provide an infinite number of control blocks, etc. Realistically, the general case can be approximated with a modest system, since the purpose of a process control system is to control a real, finite process. Recognizing that this system will be used in an academic environment, the scope of process control blocks can be expected to be less than that required of commercial units. Many available programmable process controllers are designed to handle from 1 up to 8 or 16 variables. [7] In the Process Dynamics and Control class environment, the largest experiment planned consists of two cascaded continuous stirred-tank reactors with control desired over the flow into and temperature of each. This would require 4 output lines and at least 4 input lines. A complex control strategy would look at more process state variables than just the 4 being controlled to determine what control signals to send to the respective 4 actuators. To permit maximum use of the mathematical operations possible with the system, sufficient inputs are necessary to re-

ceive several sensor signals per controlled variable. For example, to provide very accurate control of the internal temperature of a continuous stirred-tank chemical reactor, the control strategy would be based on the current temperature in the reactor, the flow rate of chemical stock into the reactor, the temperature of the stock flowing into the reactor and the flow rate of the chemical products leaving the reactor. This strategy would require 4 input channels for the sensors and 1 output channel for the reactor heater control. Different control strategies will obviously call for different combinations of inputs and outputs, although it would appear from the expected use that this 4 to 1 ratio should be provided. As a result, it would be desirable for this system to have 16 input channels to fully support its 4 output channels.

Reviewing definitive texts on process control, [1,2], confirmed that a practical process control system must have P, PI, PID functions available, a means to obtain a square-root of a process variable (to linearize certain types of variables) and a means to interconnect control loops for cascade control strategies. The literature also points to sampling rates in the neighborhood of 1 second.

While the Chemical Engineering Department at the University of Washington does not currently have experiments set up utilizing real time control of physical processes,

it does have the process equipment with the necessary sensors and actuators for use with this process control system. The equipment is currently used to demonstrate process dynamics, but feedback control is not available. These sensors provide, and the actuators utilize, proportional 0 to +5 volt D.C. signals.

Using this information, the desired specifications for this process control system were developed.

SYSTEM SPECIFICATIONS

A. Communications Channels:

1. 16 analog input channels.
2. 4 analog output channels.
3. Analog signal range 0 to +5 volts.
4. RS-232C Serial data channel to PDP-11/60 computer.

B. System Sampling Rate:

1. User specified.
2. All channels sampled at same rate.
3. Min: 0.1 sec; Max: 10.0 sec.
4. Resolution: 0.01 sec.

C. System Operating Modes:

Operating modes are selected by user from the system terminal.

1. CONFIGURATION.

a. Interacts with user to create a config-

uration file in memory which describes the desired process control system.

b. Configuration file includes file name, sample rate, inputs, outputs, definition of signal paths, set points and functional blocks used.

2. MODIFY.

Interacts with user to permit editing of a configuration file in memory.

3. RUN.

a. System configures real-time process control program according to contents of configuration file currently in memory.

b. On command, executes that program.

c. System responds to terminal input to change parameters during execution.

4. TERMINAL.

a. System emulates PDP-11/60 remote terminal.

b. Enables transfer of data and configuration files to and from PDP-11/60.

c. Enables user full access to PDP-11/60.

D. Functional Blocks:

1. Quantity: 32, any combination of types.

2. Inputs:

a. 2 per block, except SQT and SCL which have 1.

b. Permitted values:

1. Any input channel.

2. Output of any block.

3. Decimal Constant up to 5 digits maximum.

3. Types:

a. Operational: Provides mathematical operation on 2 inputs, x and y, except as noted.

1. ADD (x+y)*Gain.

2. SUB (x-y)*Gain.

3. MUL (x*y)*Gain.

4. DIV (x/y)*Gain.

5. SQT (SQRT(x))*Gain.

6. SCL (x)*Gain.

b. Control: Provides P, PI or PID control algorithm on inputs x and y. User selects which algorithm by choice of Rate and Reset parameters. The algorithm choice is shown below:

1. P Reset = 0, Rate = 0.

2. PI Reset = finite, Rate = 0.

3. PID Reset = finite, Rate = finite.

4. Gain:

a. Block gain is multiplied times the output of the function performed.

b. Decimal Constant up to 5 digits maximum.

5. Block Output Option:

   a. Each block may be optionally placed in a manual mode where its output is forced to be a constant value supplied by the user.

   b. Option is selected/rejected by yes/no entry.

   c. If yes, output value of decimal constant up to 5 digits maximum is required.

   d. If no, the normal output of the block is utilized and no constant is required.

6. PID parameters:

   a. Rate, D: From 0 to 99999 Seconds.

   b. Reset, R: From 0 to 99999 1/Sec.

E. Data Collection:

1. User may select up to 6 points in the configuration from which to collect data each sampling period. Values of these points will be displayed on the terminal in real time during Run Mode. They may also be sent to the PDP-11/60 computer and saved in a data file there for later analysis and plotting.

2. Permitted points in the data file:

   a. Any input channel.

   b. Any block output.

F. Input Connections:

1. All input channels are scanned by the system. Only

those channels named as inputs to blocks or which are connected to output channels are actually used.

2. An input channel may be used any number of times.

G. Output Connections:

1. User specifies which internal signals go to those output channels (maximum of 4) that are to be used.

2. Permitted signals:

a. Any input channel.

b. Any block output.

H. User Control:

1. User exercises control from CRT terminal.

2. User selects options from menus displayed.

Chapter 3

HARDWARE DESIGN

HARDWARE REQUIREMENTS

The requirements for hardware components and their interconnections stem from the need to implement the process control system specifications given in the previous chapter. Figure 4 offers a simplified block diagram of the complete hardware system to aid in visualizing the individual components that will be required.

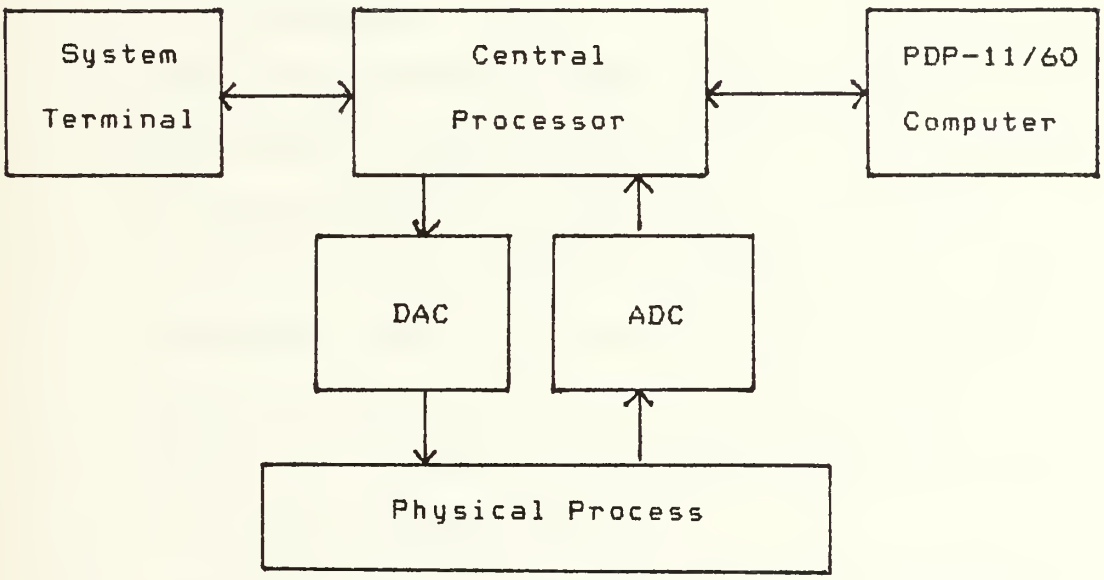


Figure 4. Process Control System, Simplified Block Diagram

Starting with the Central Processor, the following requirements can be readily identified for the hardware com-

ponents.

A. Central Processor.

The Central Processor must contain the system software in its memory and execute it to provide the system operating modes specified. This includes communication with the terminal, with the process and with a second computer. It also means the Central Processor must be able to operate in real time. To accomplish this it must have the following:

1. Interrupt processing capability and at least one programmable timer.
2. Memory capacity of at least 8K bytes of ROM to handle the estimated program size and 8K of RAM for the estimated variable data requirements.
3. Two serial port controllers (USARTs) for communication with the system terminal and for communication with the PDP-11/60 computer.
4. Capability to control and communicate on a bi-directional parallel data bus.
5. Capability for floating point operations, preferably in hardware, to handle the number of calculations necessary within the time allotted by the real time program. Particularly desired are multiply, divide, add, subtract, square-

root, integer to floating point and floating point to integer conversion.

B. Analog-to-Digital Conversion

The Analog-to-Digital Converter (ADC) in Figure 4 refers to the conversion of analog signals received from the process equipment to equivalent binary integers. The software can then convert them to floating point numbers and manipulate them in the central processor while running a real time control program. The converter must have the following characteristics:

1. Multiplex 16 analog inputs into the converter.
2. Input range of 0 to +5 volts DC full scale.
3. Provide 12 bit straight binary integer output.
4. Program controllable for channel selection and conversion start.
5. Bus compatable with central processor.

C. Digital-to-Analog Converter

The Digital-to-Analog Converter (DAC) converts the binary integers generated by the software to analog voltages between 0 and +5 volts DC that can be used by the process equipment actuators. The DAC must have the following characteristics:

1. Have 4 independent DACs, each with a 12 bit straight binary input and a 0 to +5 volt DC output.
2. Bus compatable with central processor.

D. System Terminal

A data entry terminal is necessary for user communication with the system. For the anticipated brief exchanges of prompts, system commands, menu selections and real time data a video terminal with minimal features is sufficient. It must at least have:

1. Ability to recognize full ASCII character set.
2. Full duplex capability to enable communication through the process control system to the PDP-11 in the Terminal Mode.
3. Direct cursor addressing to enable display formatting, especially during Run Mode.

E. Support Components

A chassis, power supply and connection panel will be needed to support these components. The chassis must include the bidirectional bus for the processor and other components to communicate with each other. The connection panel provides coaxial BNC connectors to permit connection

of the system to the physical process it is to control.

HARDWARE SELECTION

Of the available microprocessors, any one of the 8 or 16 bit processors would meet the hardware requirements. Depending on the processor chosen, varying amounts of additional circuitry and board design effort would be required to make the processor functional in this application. A number of manufacturers of processors also offer "single board computer" assemblies which contain the processor and the supporting hardware necessary to operate the processor almost as a stand alone unit. Such assemblies meet most of the requirements previously laid down for the process control system central processor. These single board computers are usually compatable with a bus made by the manufacturer. Additional components, such as A/D and D/A converters, RAM memory, communications port controllers, etc. are also offered by the manufacturer for use with the single board computer and the bus.

Of the many hardware combinations possible, two potential system configurations, based on the popular Intel 8085 and Motorola 6800 microprocessors, respectively, are shown in Table 1. [8,9] The components listed are board assemblies which directly plug into the manufacterer's bus, Intel's Multibus or Motorola's EXORbus.

The Intel Floating Point Processor is an exception, however. It plugs directly into a socket on the SBC 80/24 Single Board Computer. The Intel chassis provides slots, power and bus for 8 cards; the Motorola chassis provides for 10 cards.

Component	Intel	Motorola
Single Board Computer	SBC 80/24	M68MM01D
Memory Expansion (16K Dynamic RAM)	SBC 116A	MEX6816-1
Second Serial Port	(on SBC 116A)	MEX6850
Floating Point Processor	SBX 331	M68MM14
A/D Converter	SBC 711	M68MM15A
D/A Converter	SBC 724	M68MM15CV
Chassis	SBC 660	M68MMLC

Table 1. Hardware Configuration Alternatives

An important consideration in hardware selection is the amount of logic development support available. The Electrical Engineering Department of the University of Washington has extensive facilities for developing and testing Intel microprocessor systems. These include two Intel MDS 800 Microcomputer Development Systems with emula-

tors for 8080, 8085 and 8086 microprocessor systems. As a result of the availablity of these design tools, and the familiarity with their use, the Intel system is the preferable hardware choice. With this in mind Professor Ricker contacted the Intel Corporation and requested assistance in obtaining the Intel hardware components identified for this process control system. Intel graciously donated all of the components necessary, and also provided reference manuals to aid in implementing the design. The Chemical Engineering Department provided an ADM-3A video terminal for use as the system terminal, along with necessary connections between the process control system and the process equipment to be controlled.

HARDWARE DESCRIPTION

With the hardware components identified, a description of how each operates, as well as its interaction with other components is necessary to integrate the system design. Figure 5 presents a detailed block diagram of the system hardware.

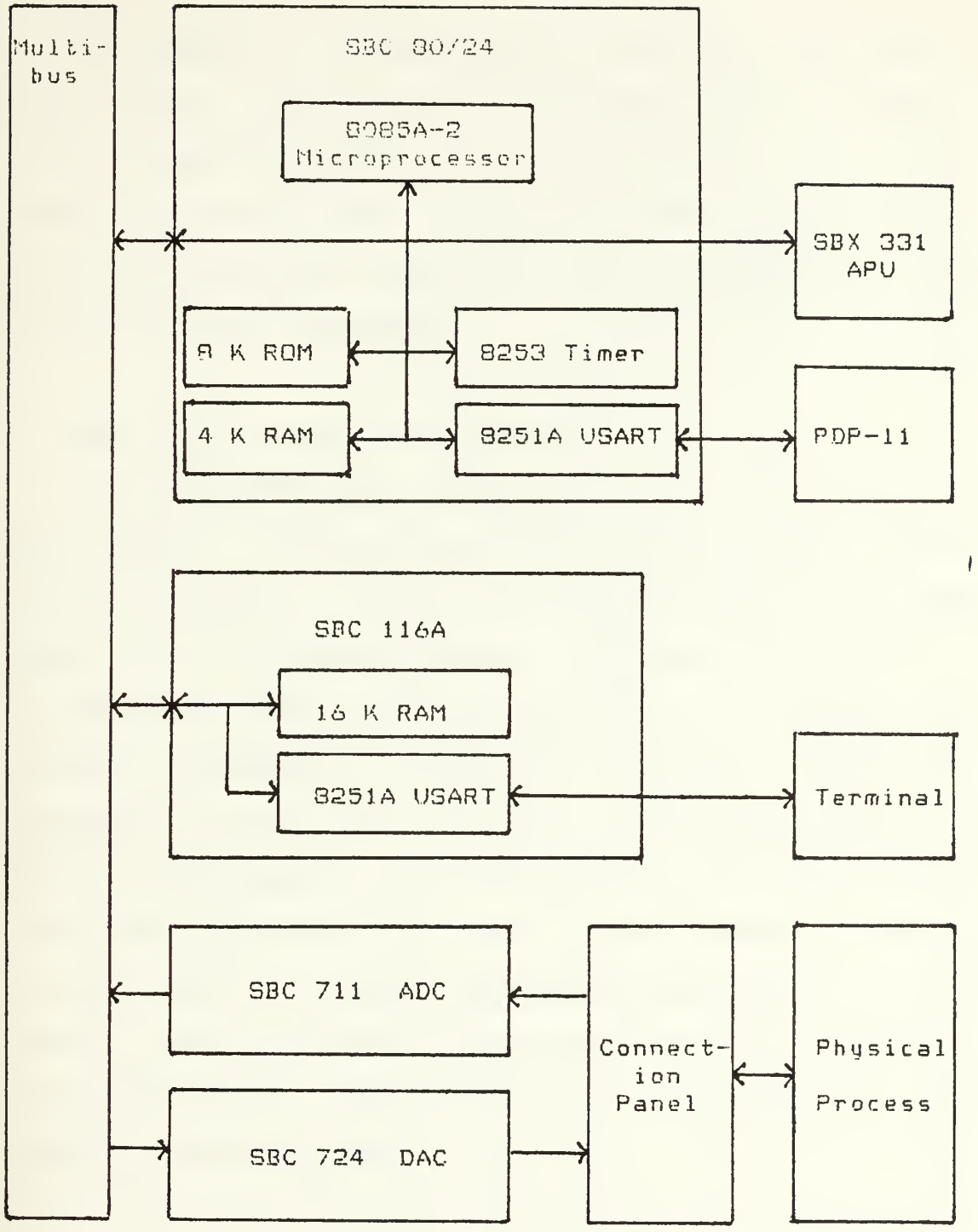


Figure 5. Process Control System, Detailed Block Diagram

Proceeding in the same order as with the requirements, the central processor will be covered first. In meeting the requirements for the central processor, three components are actually used: the single board computer, a memory and I/O expansion board and a math multimodule. Each will be covered separately.

A. SBC 80/24 Single Board Computer.

The SBC 80/24 single board computer is the heart of the system. It contains an 8085A-2 microprocessor operating at 4.84 MHZ, a serial port controlled by an 8251A USART, six 8 bit parallel ports controlled by two 8255 programmable peripheral interface devices, an 8259A eight channel programmable interrupt controller, an 8253 programmable interval timer with three independent timers, 4K bytes of RAM memory, up to 32K bytes of ROM memory and bus controller circuitry. The board also contains circuitry for the addition of up to 2 special purpose SBX multimodule boards, such as the SBX 331 Arithmetic Processor Unit, to the 80/24 board. Figure 6 provides a detailed block diagram of the 80/24 board.

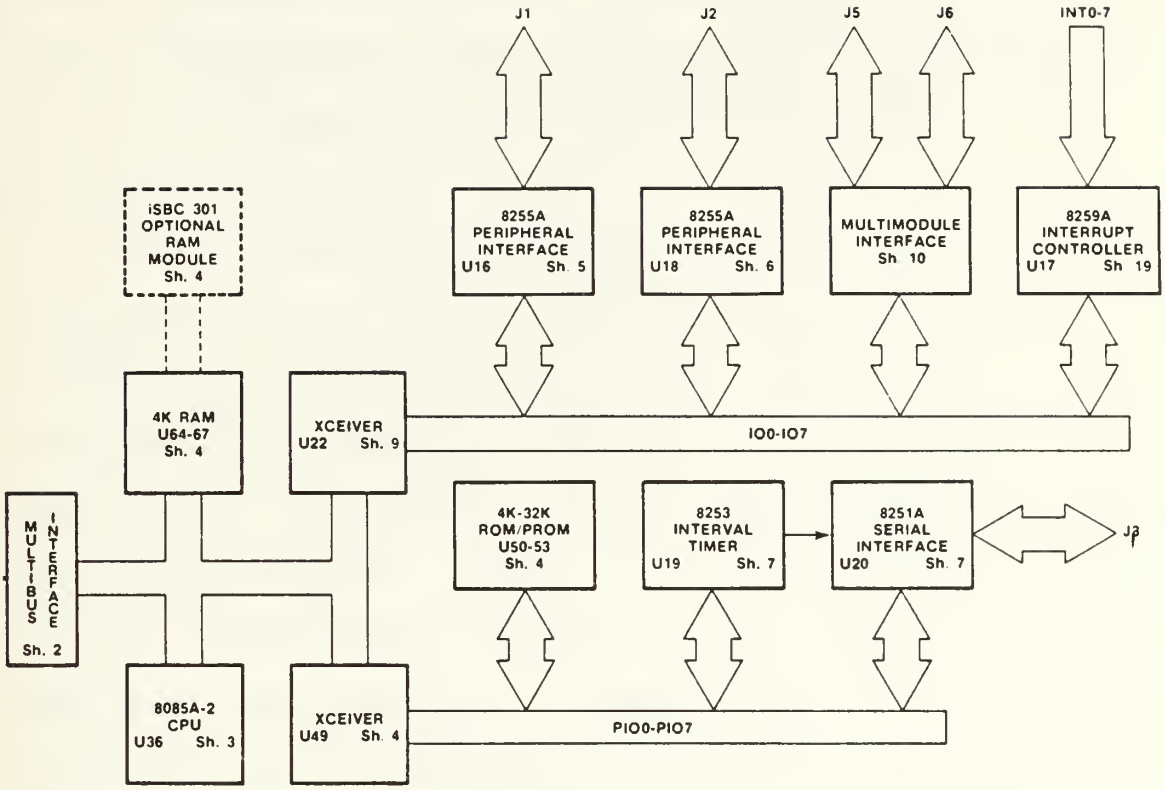


Figure 6. SBC 80/24 Single Board Computer, Block Diagram

(Intel iSBC 80/24 Single Board Computer Hardware Reference Manual, p 4-1.)

Switches and jumpers on the board determine the port numbers of on-board I/O ports and the starting addresses of RAM and ROM memory blocks. The 80/24 board decodes all references to memory and I/O ports. If the reference is to an on-board asset, the read or write operation is accomplished directly. If the reference is not recognized as on-board, the processor uses the bus control circuitry to

gain control of the Intel Multibus, and initiate a read or write operation onto the bus. When a device with the address or port number referenced recognizes the operation is for it, it acknowledges this and writes or reads data on the bus as required.

For this application the RST 7.5 interrupt is enabled on the board and is connected to the output of counter 1 of the 8253 Programmable Interval Timer. The counter is controllable from software to provide an interrupt driven real time clock. The 8251A Universal Synchronous/Asynchronous Receiver/Transmitter (USART) is used to provide an asynchronous RS-232 port at 9600 baud for communication with the PDP-11/60 computer. The 8259A Programmable Interrupt Controller and the two 8255A Programmable Parallel Interface devices are not used in this application, but are available for future enhancement of the system. [10]

B. SBC 116A Memory and I/O Expansion Board.

The SBC 116A memory and I/O expansion board provides the bulk of the RAM memory that will be available for use by the system software for storage of variable data. A functional block diagram of the board is provided in Figure 7.

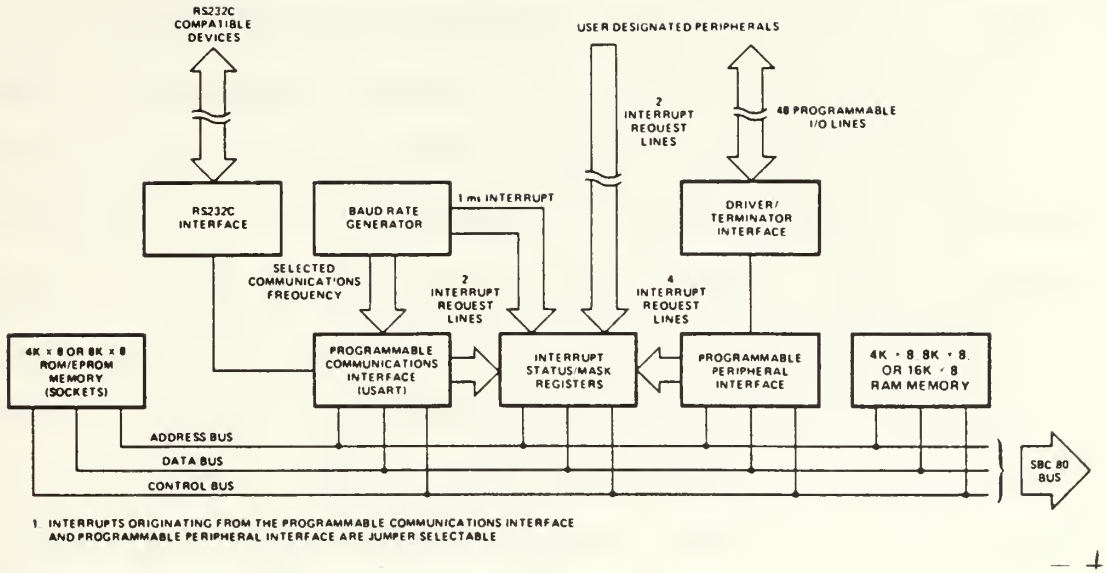


Figure 7. SBC 116A Expansion Board, Block Diagram

(Intel iSBC 104/108/116 Combination Memory and I/O Expansion Board Hardware Reference Manual, p 4-1.)

The starting address of the RAM is set on the board to immedidately follow the last address of RAM on the SBC 80/24 board, thus forming a contiguous 20 K-bytes of RAM. This board also contains an 8251A USART which will be used to provide the serial data port to the system terminal. Jumpers on the board are set to operate this serial port at 9600 baud. The two 8255A Parallel Peripheral Interfaces on the board are not used in this application. The board also has capability for adding up to 8K of ROM memory to the system, which also will not be required. [11]

C. SBX 331 Fixed/Floating Point Math Multimodule Board.

The SBX 331 Fixed/Floating Point Math board uses an 8231A Arithmetic Processor Unit (APU) and is connected directly to the single board computer via an internal bus structure. It is accessed as though it were a peripheral device through an I/O port on the single board computer. Floating point operands are output to the APU, then an op-code is output to the APU instructing it which operation it is to perform. Results are then read back from the port. The APU is a stack oriented device, that is, the operands are presented one at a time, followed by the operation to be performed, in much the same way as arithmetic operations are performed on a Hewlett-Packard hand-held calculator. In this application, the APU is polled and tested for completion of calculation vice using its interrupt capability. The APU performs the following operations in hardware: [8, 12]

1. Conversion of 16 bit binary to 32 bit floating point.
2. Conversion of 32 bit floating point to 16 bit binary.
3. Floating point multiply, divide, add and subtract.
4. Floating point sine, cosine, tangent and

inverses.

5. Floating point square root, natural and common logs and inverses.
6. 16 and 32 bit integer multiply, divide, add and subtract.

D. SBC 711 Analog Input Board.

Analog-to-digital conversion is obtained through use of the SBC 711 board. A block diagram of the SBC 711 ADC is given in Figure 8. The SBC 711 can be commanded to convert a single channel specified by software or to convert a sequence of channels starting with the first and last channel specified by software. Completion of the task can be signalled either by an interrupt or by polling the status register. In addition to straight conversion, the board provides software selectable gains of 1, 2, 4 and 8 which are applied to all analog channels before conversion to binary. Conversion is initiated by setting the appropriate channel number(s) in the channel register(s) and setting the appropriate bits in the command/status register.

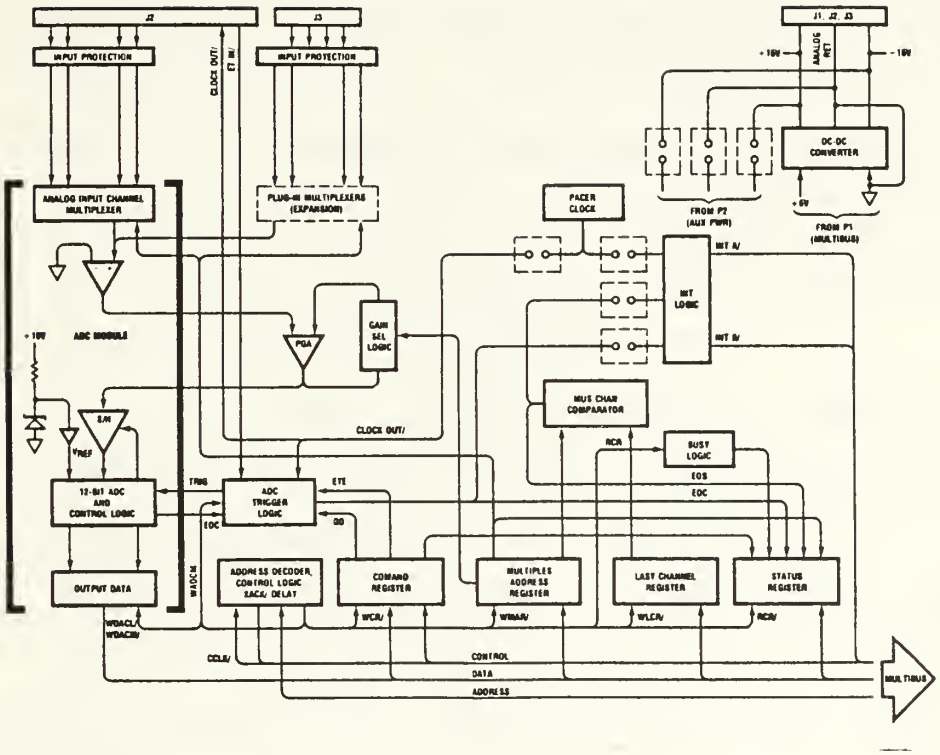


Figure 8. SBC 711 Analog Input Board, Block Diagram

(Intel iSBC 711 Analog Input Board Hardware Reference Manual, p 4-3, 4.)

Communication with the board is accomplished through memory mapped I/O. The board has a header which is wired to establish a base address. The board then recognizes memory references on the bus for that address and the subsequent five addresses as being directed to the SBC 711. The uses of the reserved addresses are as follows: [13]

1. Base + 0: Command/Status Register.

2. Base + 1: Channel Number and Gain Register. For single channel conversions, the channel to read is placed here. For conversion of a sequence of channels, the number of the first channel is placed here, in the lower 5 bits. The upper 2 bits select the gain of the converter: 1, 2, 4 or 8 x the input.
3. Base + 2: Last Channel Register. Used to identify the last channel number when a sequence of channels is to be converted.
4. Base + 3: Clear Interrupts. When using the end of conversion interrupt feature, writing to bits 4 and 5 in this address will clear interrupts and reenable them for future conversions.
5. Base + 4: High Data Byte. When conversion of a channel is complete, contains most significant 8 bits of result.
6. Base + 5: Low Data Byte. When conversion complete contains lowest 4 bits of 12 bit result, left justified and filled with 0's.

E. SBC 724 Analog Output Board.

Digital-to-analog conversion is obtained through use of the SBC 724 board. This board contains 4 separate and independent digital-to-analog converters (DACs), each of

which converts a 12 bit integer to a 0 to +5 VDC output. Figure 9 contains the block diagram of the SBC 724 board.

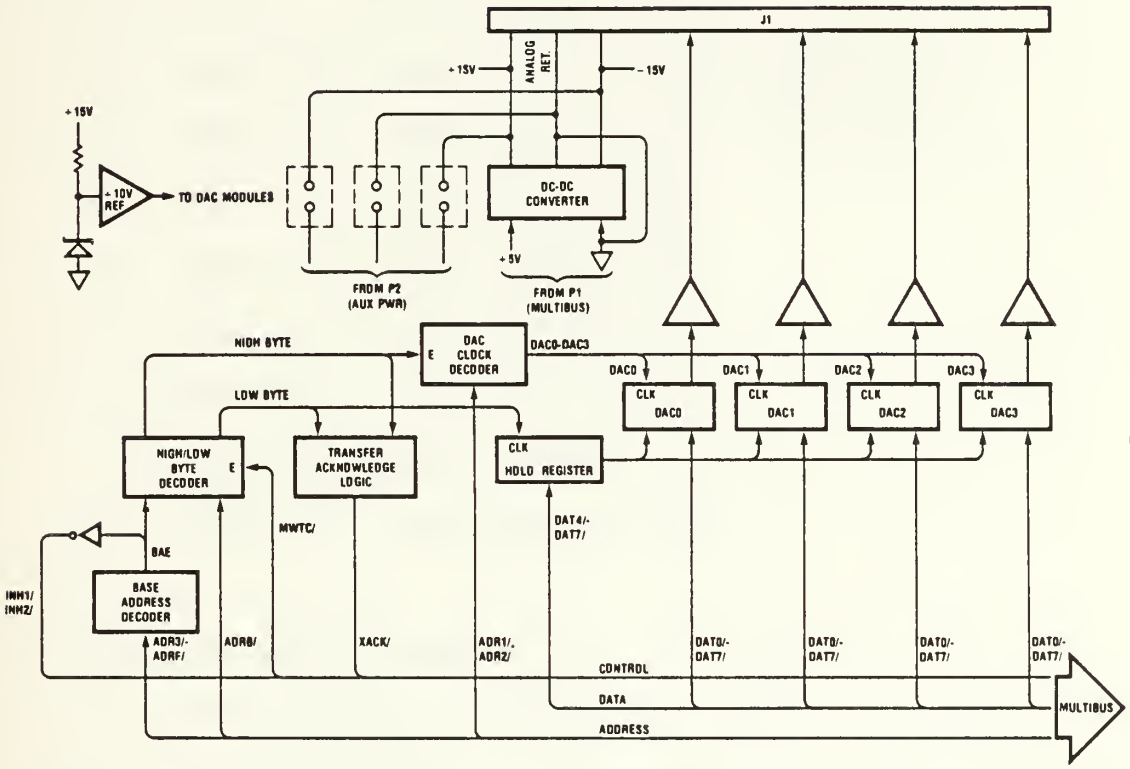


Figure 9. SBC 724 Analog Output Board

(Intel iSBC 724 Analog Output Board Hardware Reference Manual, p 4-2.)

The SBC 724 board is also a memory mapped I/O device. The board has a header which is wired to select a base address. The board then recognizes that address and the subsequent seven addresses when they appear on the bus as references to the board. Each DAC is accessed by FIRST writing the low byte to its lower byte address, THEN writ-

ing the high byte to its upper byte address. No commands are necessary to start the conversion. The addresses of the DAC's in terms of the base address are: [14]

1. Base + 0: Lower byte, DAC 0 (LS 4 bits)
2. Base + 1: Upper byte, DAC 0 (MS 8 bits)
3. Base + 2: Lower byte, DAC 1
4. Base + 3: Upper byte, DAC 1
5. Base + 4: Lower byte, DAC 2
6. Base + 5: Upper byte, DAC 2
7. Base + 6: Lower byte, DAC 3
8. Base + 7: Upper byte, DAC 3

F. SBC 660 System Chassis.

The SBC 660 System Chassis provides an eight slot cardcage and backplane for use with SBC single board computers and expansion boards. It contains the power supply for the system and dual cooling fans. The backplane provides the interconnections that make up the Intel Multibus and distribute power to the boards. The backplane has been jumpered to provide slot (2) with the highest priority in obtaining control of the bus. The SBC 80/24 single board computer is placed in this slot. This permits space for the SBX 331 which is connected on top of the SBC 80/24 board. [15]

HARDWARE CONNECTION DATA

This section contains the table listings which identify the I/O port assignments, memory address assignments, connector pin assignments which are used to permit the hardware components to operate together as a system. Table 2 identifies the I/O port assignments. In addition to those functions being used in this application, unused functions which have hardwired dedicated port assignments are also listed to show those ports as unavailable for other use.

I/O Port	Location	Function
B0 to B3	SBC 80/24	External Interrupt Expansion (not used)
B4 to B7	SBC 116A	PPI Number 3 (not used)
B8 to BB	SBC 116A	PPI Number 4 (not used)
BC	SBC 116A	System Terminal Data Port
BD	SBC 116A	System terminal Control Port
C0 to CF	SBC 80/24	SBX 331 APU
D8 to DB	SBC 80/24	External Interrupts (not used)
DC	SBC 80/24	Timer 0
DD	SBC 80/24	Timer 1 (Real Time Clock)
DE	SBC 80/24	Timer 2 (Port EC,ED baud rate)
DF	SBC 80/24	Timer Control Register
E4 to E7	SBC 80/24	PPI Number 1 (not used)
E8 to EB	SBC 80/24	PPI Number 2 (not used)
EC	SBC 80/24	PDP-11 Link Data Port
ED	SBC 80/24	PDP-11 Link Control Port

I/O port numbers are hexadecimal.

I/O port address space is from 00 to FF.

Table 2. Input/Output Port Assignments

Table 3 lists the assignments of RAM and ROM memory addresses which the system will be using.

Address	Assignment
0000	ROM start, System Software
2FFF	ROM max end
3000	RAM start
3FFF	last RAM on 80/24
4000	first RAM on 116A
7FFF	RAM end on 116A
8000 TO F6FF	Unused
F700 to F705	SBC 711 ADC
F708 to F70F	SBC 724 DAC
F710 to FFFF	Unused

Memory Addresses are hexadecimal.

Memory Address space is from 0000 to FFFF.

Table 3. Process Control System, Address Assignments

Table 4 lists the individual signals entering the system from the physical process and their connector pin assignments. The signals are received via two-conductor pairs. The signal itself is labeled "In" and the analog return or ground for the signal is labeled "Ret" in Table 4. Signals coming into the process control system are received at the system connection panel at BNC coaxial connectors S00 through S15. They are transferred directly from the connection panel via flat ribbon cable to connec-

tor J2 on the SBC 711 Analog Input Board. Table 5 lists the output signals from the system to the process under control and their connections. As with the analog inputs, each output is carried via two conductors. In Table 5, the signal is labeled "Out" and its analog return is labeled "Ret." Outputs from the system originate at connector J1 on SBC 724 Analog Output Board and are transferred via flat ribbon cable to BNC coaxial connectors C00 through C03 on the system connection panel. Figure 10 shows the system connection panel coaxial connectors.

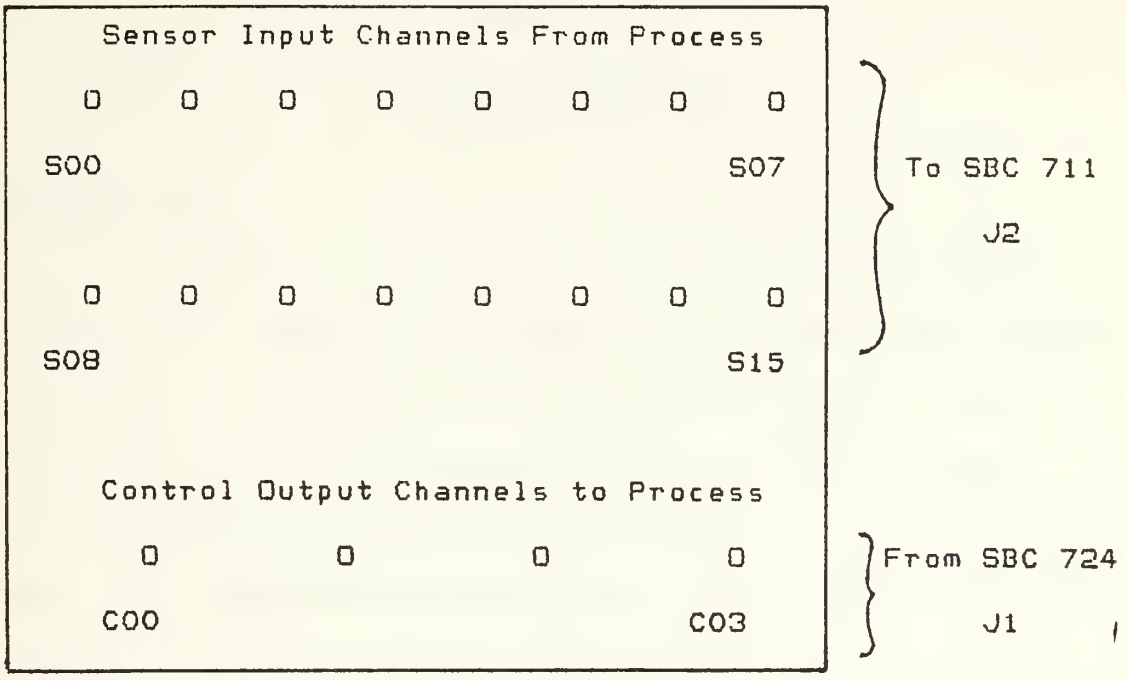
The connection to the system terminal is from card edge connector J3 on the SBC 116A Memory and I/O Expansion board via flat ribbon cable to the female RS-232 connector on the rear of the chassis labeled "terminal." The connection to the PDP-11/60 computer is from card edge connector J3 on the SBC 80/24 Single Board Computer via flat ribbon cable to the female RS-232 connector on the rear of the chassis labeled "PDP-11."

Signal In	SBC 711 J3 Pin	BNC Connector
CH 0 In	4	S00
CH 0 Ret	3	
CH 1 In	8	S01
CH 1 Ret	7	
CH 2 In	12	S02
CH 2 Ret	11	
CH 3 In	16	S03
CH 3 Ret	15	
CH 4 In	20	S04
CH 4 Ret	19	
CH 5 In	24	S05
CH 5 Ret	23	
CH 6 In	28	S06
CH 6 Ret	27	
CH 7 In	32	S07
CH 7 Ret	31	
CH 8 In	6	S08
CH 8 Ret	5	
CH 9 In	10	S09
CH 9 Ret	9	
CH 10 In	14	S10
CH 10 Ret	13	
CH 11 In	18	S11
CH 11 Ret	17	
CH 12 In	22	S12
CH 12 Ret	21	
CH 13 In	26	S13
CH 13 Ret	25	
CH 14 In	30	S14
CH 14 Ret	29	
CH 15 In	34	S15
CH 15 Ret	33	

Table 4. Process Control System, Input Connections

Signal Out	SBC 724 J1 Pin	BNC Connector
CH 0 Out	42	CO0
CH 0 Ret	45	
CH 1 Out	36	CO1
CH 1 Ret	39	
CH 2 Out	30	CO2
CH 2 Ret	33	
CH 3 Out	24	CO3
CH 3 Ret	27	

Table 5. Process Control System, Output Connections



O = Coaxial Connector.

Figure 10. System Connection Panel

Chapter 4

SOFTWARE DESIGN

INTRODUCTION

The process control system software is designed to be a modular, hierarchial, table driven real time program. The system software acts as its own operating system and functions as a self-contained stand alone program. The software interacts with the user through a video terminal, with the physical process through addressable A/D and D/A conversion boards and with a PDP-11/60 computer through an RS-232 serial port. The software is resident in Read Only Memory and begins execution immediately upon providing power to the system.

A User's Manual for the process control system is provided in Appendix A.

SOFTWARE ORGANIZATION

The software is organized into 4 levels. The highest level contains the system executive routine which initializes the system and controls execution of system commands. The system commands are short mnemonic expressions which the user enters from the system terminal to inform the executive which system operating mode the user desires. The next highest level in the hierarchy contains the system

routines, each of which implements a system operating mode. The third level of routines contains function routines which perform specific major tasks peculiar to an operating mode. As an example, in the file creation mode there is a function routine, LDBLCK, which performs the steps required to enter and check all information neccessary to specify one block in the configuration. The last level in the hierarchy contains the subroutines which provide the bulk of the actual instruction execution. Subroutines are public and may be called by routines at any level.

The relationship of the executive routine, named PCSEXC, and the three systems routines, named PCSCFG, PCSRUN and PCSPDP, are shown graphically in Figure 11. PCSCFG enables creating and editing system configuration files. PCSRUN enables executing a configuration file as a real time control program. PCSPDP enables the user to communicate directly with the PDP-11/60 computer from the process control system video terminal.

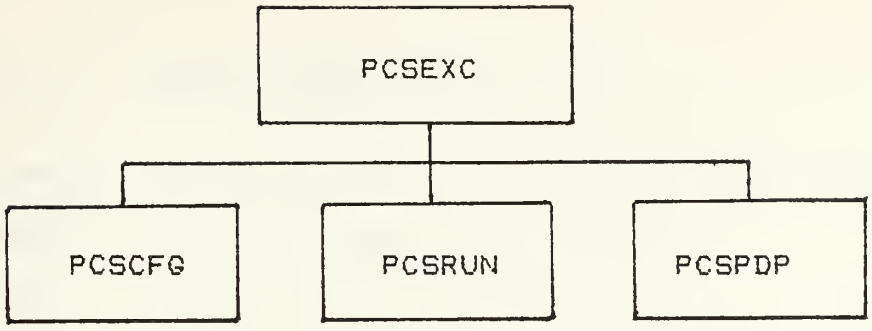


Figure 11. Software Organization, System Routines

Before describing the various routines in detail, a discussion of the data structures which they use is needed. Two primary structures are employed by the system software. The first is the configuration file. This file contains all of the information needed to describe the configuration of the desired process control system. The file is organized as an indexed sequential file of fixed record length. The file is also structured as a text file, that is, each record is one line of characters and is terminated with a carriage return character. This organization was chosen over pure sequential and random access structure to minimize file handling overhead and still provide reasonably fast access to records. Table 6 demonstrates the file structure.

Index	Record Description
CFNAME	Configuration file name, 32 characters
DFNAME	Data file Name, 32 characters
PRDREC	Sampling period, 8 characters
BLKCTR	Number of blocks in configuration
TREC	32 Block type records, each 8 ch.
XREC	32 Block X input records, each 8 ch.
YREC	32 Block Y input records, each 8 ch.
GREC	32 Block gain records, each 8 ch.
OREC	32 Block output constant option records
VREC	32 Block output constant value records
RREC	32 Block reset records, each 8 ch.
DREC	32 Block rate records, each 8 ch.
OUTCTR	Number of configuration output channels
OUTREC	4 Output channel records, each 8 ch.
DATCTR	Number of data points in configuration
DATREC	6 Data point records, each 8 ch.
ENDREC	Zero byte end of file record

Table 6. Configuration File Structure

The second principal data structure consists of the run time tables. These are the tables used by the system to implement the desired control strategy. Prior to exe-

cuting the control operation, the system translates the configuration file information to floating point numbers, internal addresses and flags which are stored in a series of tables. Then, during the running of the control operation, the software refers to the tables to determine what actions it must take. Table 7 illustrates the organization of these tables.

Table	Contents
TTBL	32 Block type codes, each 1 byte
XTBL	32 Block X input addresses
YTBL	32 Block Y input addresses
KTBL	32 Block X or Y constants, each 4 bytes
GTBL	32 Block gains, each 4 bytes
OTBL	32 Block output flags, each 1 byte
VTBL	32 Block output constants, each 4 bytes
RTBL	32 Block reset constants, each 4 bytes
DTBL	32 Block rate constants, each 4 bytes
LTBL	32 Block previous inputs, each 4 bytes
CTBL	4 Output channel addresses
PTBL	6 Data point addresses

Table 7. Run Time Tables

The organization of the system routines will be handled individually to show the relationships of the function routines each uses to implement its respective system operating mode.

The first system routine, PCSCFG, implements two commands, FILE and MOD, to permit both creating and modifying a configuration file. The executive passes a flag, named Edit, to signal which command the routine is to execute. PCSCFG uses 5 function routines to load configuration data into the file. These routines are LDNAME, LDPERD, LDBLCK, LDOUT and LDDATA. They input, check and store the information for file name, sample period, block definition, output channel definition and data point definition, respectively. Their relationship is shown in Figure 12.

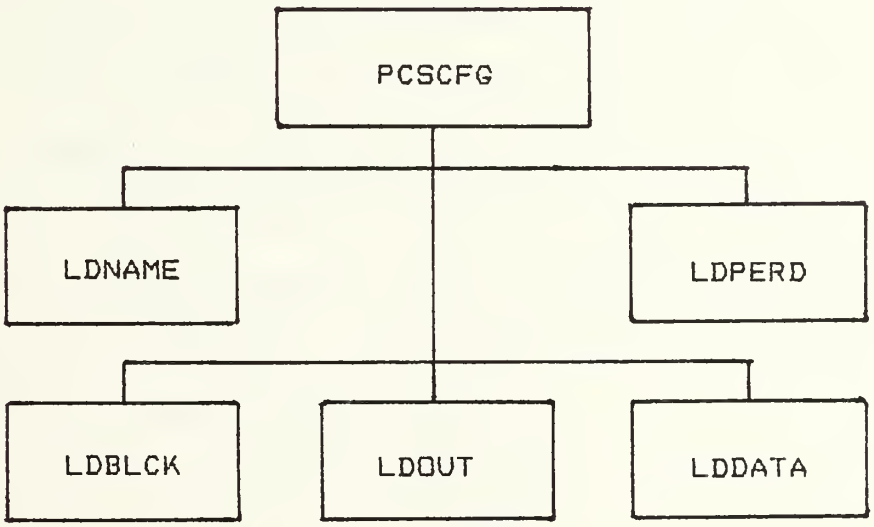
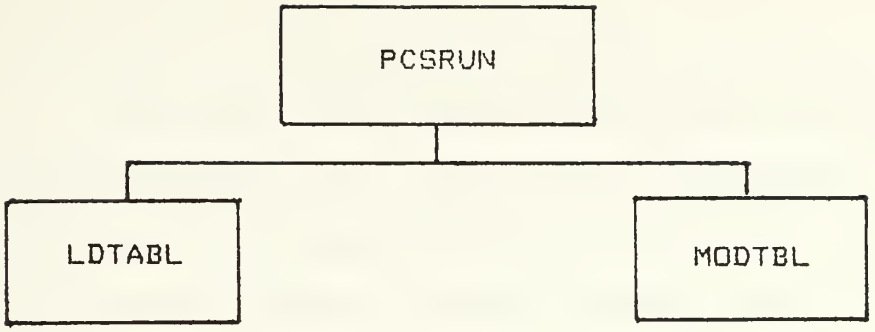
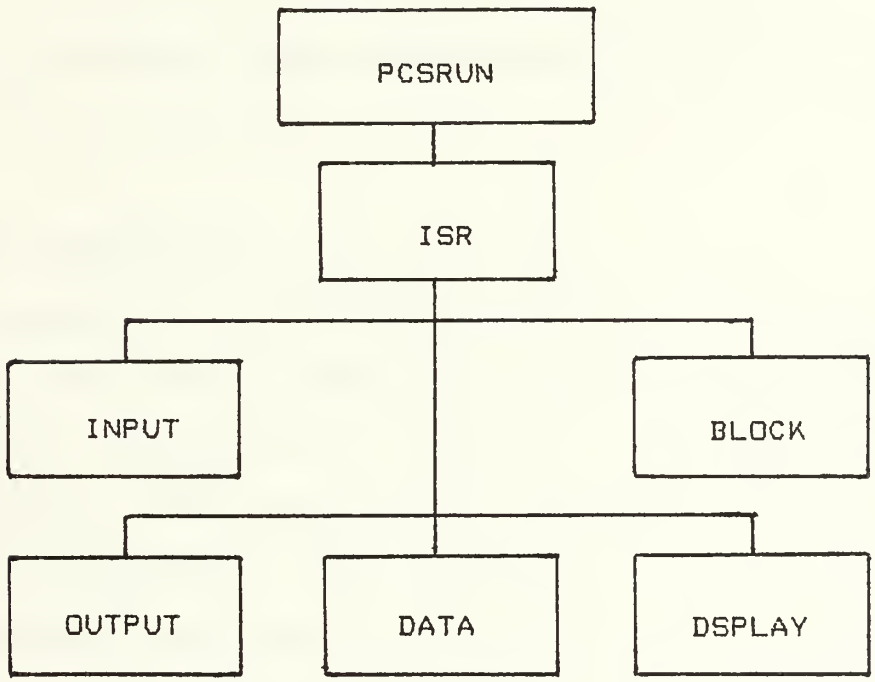


Figure 12. Software Organization, Configuration Functions

PCSRUN implements the RUN command. It takes the information describing the process control configuration from the file, converts it to table entries and executes its real time control program. PCSRUN is organized into two priority levels of routines. The lower level consists of two functions, LDTABL, which does the conversion and storing of table entries and MODTBL, which communicates with the user to enable modifying the values of constants in the tables. The higher priority level consists of the function routines which are a part of the interrupt service routine, ISR. These consist of INPUT, BLOCK, OUTPUT, DATA and DSPLAY. When a timing interrupt is received from the hardware, the low priority function MODTBL is suspended and ISR immediately begins execution. When it has completed its tasks, it returns execution to MODTBL which resumes at the point where it was interrupted. The ISR functions form the real time control program, and involve sampling the process sensors, servicing the functional blocks, delivering control output signals to the process, sending process data to the PDP-11 computer and displaying the process data on the system terminal. The relationship of PCSRUN and its routines are shown in Figure 13.



a. Low Priority Functions



b. High Priority Functions

Figure 13. Software Organization, Run Functions

PCSPDP implements the system command PDP to enable two way communication between the user and the PDP-11/60 com-

puter. It passes each character received from one to the other. In addition, the user may instruct the PDP-11 to run a program resident with its system, called $PCS, which will enable direct exchange of configuration files and data between the process control system and the PDP-11. These tasks are relatively simple and have been accomplished with several subroutines. Since no function routines were required, no organizational chart is presented here.

More detailed discussion of these routines is contained in the next section.

SOFTWARE DESCRIPTION

A number of points concerning the software routines in general need to be made before describing any of them in particular. Because of the nature of the application, a stand alone real time process controller, and also due to the logic development tools available, assembly language was chosen as the language in which all routines were written. Without an existing operating system to handle the variable assignments, fixed hardware addresses and I/O channels, assembly language drivers would be required for much of the I/O features, even if other portions could be written in a high level language. Secondly, to use the timing functions available in the hardware, assembly language routines are required to implement much of the

real time application. And finally, most assembly language drivers were already available for use with the hardware floating point processor. Very little remained that could effectively be written in a high level language.

To make the assembly language routines as understandable as possible, they are generally very short and have a name that comes as close as 6 characters permit to describing the task they perform. In addition, several conventions are employed to make it easier to understand the operation of the routine and to facilitate later modifications to the software.

In particular, parameter passing between routines is standardized. Parameter passing is done primarily at the subroutine level where actual data or addresses are manipulated. When a data byte is passed to a called subroutine, it is passed in the C register. When a data byte is returned to the calling routine it is returned in the A register. When an address or a 2 byte data value is passed, in either direction, it is passed in the H,L register pair. If a second address or 2 byte value is passed at the same time, it is placed in the D,E register pair. For example, the subroutine POINT is used frequently to locate a specific entry in the configuration file or in a run time table. It is given the start address of a group of records or a table in H,L, the address of a counter to use for counting

up to the location of the entry in D,E and the length of the records or entries it is counting in C. POINT returns the address which points to the desired record or table entry in H,L. Most subroutines, however, only pass 1 parameter.

Routines that operate on a file record or a global variable return a "Good/No Good" parameter to inform the calling routine that it was or was not successful in performing its task. The calling routine tests this condition and acts accordingly. The convention used sets the A register equal to zero if successful and sets A not equal to zero if unsuccessful. This is used extensively in the routines handling configuration file information.

A. The Executive System Routine PCSEXC

The system executive initializes the software and hardware when the power is first applied. It then displays the system menu. The system menu as seen by the user is shown in Figure 14.

```
PCS V1.1
PLEASE ENTER SYSTEM COMMAND DESIRED, FOLLOWED BY <CR>
FILE    CREATE PROCESS CONTROL CONFIGURATION FILE
MOD     MODIFY CURRENTLY DEFINED CONFIGURATION FILE
PDP     ACCESS PDP-11/60 AS A REMOTE TERMINAL
RUN     RUN CONFIGURATION FILE AS A REAL TIME PROGRAM
EXIT    STOP FUNCTION AND RETURN TO HERE
*
```

Figure 14. System Menu

The system commands displayed are used by the system to accomplish the modes of operation of the system. With the system menu displayed, the executive waits for an input from the user. When a line of input is received, the executive examines it and if it contains a legal system command the executive transfers execution to the system routine designed to provide the mode of operation desired. In the case FILE or MOD is entered, PCSEXC clears or sets the Edit flag, respectively, prior to calling PCSCFG. When each system routine completes its operation it returns control to the executive which then restores the system menu to the display and waits for another command input.

PCSEXC calls the following system routines:

1. PCSCFG: With Edit flag clear, implements FILE command. Cleans out configuration file buffer, outputs questions and prompts to guide user, and inputs user selections to create new file. With Edit flag set, implements MOD command. Enables user to modify contents of configuration file currently in file buffer.
2. PCSRUN: Implements RUN command. Enables running a configuration currently contained in the file buffer as a real time process control program. Enables user to change values of set points, block gains, PID reset and rate, and to remove or add constant output feature to blocks while program is running.
3. PCSPDP: Implements PDP command. Enables user to communicate directly with PDP-11 computer. System terminal becomes a remote terminal of PDP-11 system.

PCSEXC uses the following subroutines to set up the serial ports, to display information on the system terminal and to receive input from the user:

1. IUSART: Initializes the serial port control-

ers (8251A USARTS) for the system terminal and the PDP-11 link.

2. INLINE: Inputs one line of characters from the terminal. Provides correction capability using the Shift/Rubout keys. The maximum number of characters input is 79. A carriage return <CR>, a line feed <LF> or an escape <ESC> will be accepted as an end of line character.
3. DECODE: Examines a line of input for system command. If one is found, the first character of the command is returned.
4. OUTMSG: Displays a message on the system terminal.
5. CLRCRT: Clears the system terminal display.

B. Configuration File System Routine PCSCFG

The configuration file system routine, PCSCFG, accomplishes two modes of operation, creating a configuration file, and modifying a configuration file. One routine was selected rather than two to minimize the duplication in overhead keeping track of position counters and condition flags. The difference in the two modes rests primarily on the fact that in the modify mode the user has the option of either inputting new elements in the configuration description or accepting the current elements. To accomplish

this, the routine tests a flag, called Edit, at each input. If the flag is set, the old information is retrieved from the file and displayed. A carriage return is accepted as indication that the old is to be retained. If new input is provided by the user it is checked and placed in the file, overwriting the old information.

The configuration file is created, or modified, in sections. The sections are:

1. Configuration File Name Definition: Optional. User may provide 32 character maximum name to identify configuration.
2. Data File Name Definition: Optional. User may provide 32 character maximum name to identify file where he intends to store run time data.
3. Sampling Period Definition: User selects period between control system updates.
4. Block Definitions: User defines inputs and parameters for each block to be used in configuration. System presents blocks sequentially for definition, starting with the first block, B00, until either the configuration is complete or the last block, B31 has been used.
5. Output Channel Definitions: User selects the points in configuration to connect to output

channels as control signals for process. System presents channels sequentially for definition, starting with C00, and ending either when all channels needed have been defined or the last channel, C03 has been used.

6. Data Point Definitions: User selects points in configuration to sample and save as data each period. System presents data points sequentially, starting with point D00 and ending when either all points required have been defined or the last point, D05, has been used.

The block definition section is fairly involved since for each block a number of parameters as well as the block inputs are needed to fully define the block. PCSCFG uses a routine called LDBLCK to handle the questions, answers, checks and storage tasks required to define a single block. An example of the system terminal display seen by the user during definition of a block is shown in Figure 15. Entries typed by the user are shown in parentheses. In the example, the user has decided to add a constant, such as an offset, to a sensor input from the process and pass the sum on with unity gain. He has not selected the forced constant output, or manual condition, for this block.

```
FUNCTIONAL BLOCK DEFINITION,     MAXIMUM 32 BLOCKS
WHEN NO MORE BLOCKS DESIRED, ENTER <ESC> FOR TYPE
BLOCK NUMBER = B00
TYPE?  ADD,SUB,MUL,DIV,SQT,SCL OR PID <CR>
*(ADD <CR>)
X INPUT?  S00-S15,B00-B31 OR MAX 5 DIGIT CONSTANT <CR>
*(S05 <CR>)
Y INPUT?  S00-S15,B00-B31 OR MAX 5 DIGIT CONSTANT <CR>
*(0.25 <CR>)
GAIN?  MAX 5 DIGIT CONSTANT <CR>
*(1 <CR>)
CONSTANT OUTPUT?  YES OR NO <CR>
*(NO <CR>)


User input is shown in ( ).
```

Figure 15. Configuration File Routine, Defining a Block

When all of the blocks needed have been defined the user enters an escape character <ESC> instead of a type selection. The system responds by initiating the next section.

The fourth section, defining the system output channels invloves asking the user to name the point within the configuration he desires to be output to the process through the particular channel. An example of an output channel definition is given in Figure 16.

```
OUTPUT CHANNEL DEFINITION, MAXIMUM 4 CHANNELS
WHEN NO MORE CHANNELS DESIRED,     ENTER <ESC>

OUTPUT C00=?  INPUT S00-S15 OR BLOCK B00-B31 <CR>
*(B00 <CR>)

OUTPUT C01=?  INPUT S00-S15 OR BLOCK B00-B31 <CR>
*(<ESC>)


User input is shown in ( ).
```

Figure 16. Configuration File Routine, Defining an Output

The fifth section, defining data points, is virtually identical to the previous section.

After definition of the data points, the configuration file is complete. The system displays a summary of the file to enable the user to review it prior to exiting from the mode. To exit, the user enters the "EXIT" system command which returns control to the executive.

In modifying a configuration file, the routine follows the same steps as in creating the file. There are two differences, however. First, after every question displayed by the system, the answer currently in the file is also displayed. Secondly, the routine will now accept several skip codes to facilitate reaching the file entries to change. The skip codes used are:

1. <CR>: The answer displayed is to be retained,

skip to the next question.

2. <LF>: The line feed skips to the next block, leaving all remaining elements of the current block definition intact.
3. <ESC>: The escape character skips to the next section. All remaining elements in the current section are left intact.

If, when modifying a file, more blocks output channels or data points are desired than originally specified, PCSCFG automatically shifts into File Mode for that section. New blocks, channels or points are added, starting with the number next in line after the last one currently used in the file.

The function routines that PCSCFG calls are:

1. LDNAME: Inputs name, if provided, checks for 32 character maximum length and places in file. PCSCFG instructs routine which record, CFNAME or DFNAME, is to receive the name.
2. LDPERD: Inputs sample period and checks for 0.1 and 10.0 limits. Stores in file record PRDREC.
3. LDBLCK: Inputs, checks and stores each parame-

ter neededtocompletely specify a block. File record BLKCTR is updated for each block entered to enable the routine to keep track of block number. Block type, X and Y inputs, gain, output constant option and when needed, PID parameters reset and rate, are stored in file records.

4. LDOUT: Inputs, checks and stores in file records, starting at index OUTREC, the name of the signal to send out on each channel. The file record OUTCTR is updated as each channel is defined to keep track of the channels used.
5. LDDATA: Inputs, checks and stores in file records, starting at index DATREC, the name of the point in the configuration desired to be saved as data and displayed. The file record DATCTR is updated with each point definition to keep track of the points used.

C. Run Configuration System Routine PCSRUN

The system routine PCSRUN accomplishes the Run Mode of the process control system. It handles the conversion of configuration file records describing the desired control system to entries in the run time tables which will be used to actually control the physical process. When the tables

have been loaded, the routine opens communication with the user, requesting a start command. The user starts execution of the real time program by entering the command "GO" followed by <CR>. When directed to start, PCSRUN enables the 1 msec timing interrupts and waits for further input from the user.

While the process control program is running, the user may change the values of constants used in the configuration. Specifically, the user can change the gain of any block, the reset and rate of any PID block, the value of any set point given as the input to a block and he can change any block from the output held constant condition to the calculated output condition or the reverse. Changes are accomplished by entering the block number. The routine then asks for the parameter to be identified and then for the new value. After each change is made, the routine is ready to make another. The change is effected on the next timing interrupt. The execution of the control program may be stopped at any time by entering the command "HALT" followed by <CR>. The program may be then be resumed by entering "GO" or the mode may be exited by entering the system command "EXIT."

An example of the system terminal display during execution of a real time program is shown in Figure 17.

```
RUN PROCESS CONTROL CONFIGURATION
TIME 02:37


DO0 =  3.3234
DO1 = -.00005
DO2 =  1.0000


ENTER HALT <CR> TO STOP
ENTER BLOCK NUMBER <CR> TO CHANGE BLOCK
*
```

Figure 17. Run Configuration Routine, Real Time Display

The control of the process is accomplished by the interrupt service routine, ISR. ISR initiates execution upon receipt of each 1 msec timing interrupt from the hardware counter. ISR counts occurrances of the interrupts with two counters, DCNTR and PCNTR. PCNTR counts up to the value of the sampling period, in msec, to signal the need to update the process control action. DCNTR counts up to 1 second, signalling the need to refresh the real time display. If neither counter has reached its final value upon counting the current interrupt, ISR returns without further action. ISR saves the status of all hardware registers and flags to preserve the integrity of the task interrupted and to make ISR reentrant.

When PCNTR equals the period, ISR scans all 16 system input channels and stores floating point representations of

the 16 voltages read in the input table STBL. It then performs the block calculations required, starting with block B00 and continuing until the last block specified in the configuration file. ISR then connects the requested system channel inputs and/or block outputs to the system output channels. Any unused channels are forced to be zero volts DC. If a data file had been opened prior to entering the Run Mode, ISR would send ASCII character representations of the data points requested to the PDP-11 computer file Otherise, ISR returns.

When PCNTR equals 1 second, ISR updates the real time display, incrementing the run time and replacing the data point values with their current values. ISR then returns.

To accomplish these tasks, PCSRUN calls the following function routines:

1. LDTABL: Clears all tables, fetches and converts record data from the configuration file and stores it in run time tables. Three letter block types are made 1 byte hex numbers, Sxx and Bxx symbols are made absolute addresses and constant strings are made 4 byte floating point numbers.
2. MODTBL: Inputs and decodes block number, parameter code and new parameter value. Enters

new parameter value in table. Displays prompts to aid in making changes.

PCSRUN calls the following function routines through the interrupt service routine ISR:

1. INPUT: Scans all 16 analog system input channels, stores 2 byte integer values in the input buffer ADCBUF, converts the integers to floating point representation, scales them from millivolts (value of LSB) to volts and stores the final values in the input table, STBL.
2. BLOCK: Fetches each block type code, calls a routine to perform the appropriate operation and places the result in the block output table, BTBL. It then checks the output constant table, OTBL, to determine if the flag for that block is set. If it is, BLOCK overwrites the value just entered with the desired constant output value from table VTBL. BLOCK services the blocks sequentially and returns after completing the last block specified in the configuration file.
3. OUTPUT: Fetches a floating point value pointed to by the absolute address in the output chan-

nel table, CTBL. It converts the value to a 12 bit integer left justified in 2 bytes and sends the two byte value to the output channel digital-to-analog converter. All four system output channels are serviced. LDTABL initializes CTBL with the address of floating point zero, FZERO, in all four entries. The channels specified in the configuration file have this address overwritten with the absolute address of the signal desired by the user.

4. DATA: Fetches a floating point value pointed to by absolute address in data point table, PTBL. It converts this number to a 5 digit ASCII string with sign and decimal point. This value is sent to the PDP-11 computer with a trailing space character. Each specified data point is sent sequentially until the table is exhausted. After the last point, a <CR> is sent to signal the end of a sample period.
5. DSPLAY: Updates the real time clock and displays the new time, in minutes and seconds, on the terminal. DSPLAY also displays the current values of the data points on the terminal, one per line, next to the data point symbol Dxx.

D. Remote Terminal System Routine PCSPDP

The remote terminal system routine, PCSPDP, performs the Terminal Mode of Operation. This mode is selected by the system command PDP. In this mode, PCSPDP polls the two serial port controllers (8251A USARTS) looking for the status bit RX RDY set, indicating that a character has been received on the port. The character is read from the active port, examined, and if not an <ESC>, transmitted out the other port. An <ESC> received from the system terminal signals the end of the Terminal Mode session. The character is not transmitted and PCSPDP returns control to the executive. An <ESC> received from the PDP-11 signals the process control system that the previous character from the PDP-11 was a protocol character. The previous character is removed from the display, since it was not intended for the user, and is decoded. PCSPDP then performs the data transfer action required by the protocol character. These are simple tasks and are performed by subroutines called directly by PCSPDP. The protocol characters used between the two computers strictly involve file and data transfers related to the process control system. They are listed here for reference:

1. C <ESC>: Send configuration file from PCS memory to the PDP-11. This character is sent by

the PDP-11.

2. D <ESC>: Data file has been opened on PDP-11 system for input. PCS may now send data to PDP-11. This character is sent by PDP-11. It results in DFOPEN flag being set for use by PCSRUN.

3. G <ESC>: Configuration file follows immediately from PDP-11. This character is sent by PDP-11. It results in the down loading of a configuration file into the PCS configuration file buffer.

4. H <ESC>: Data file follows immediately from PDP-11. The PDP-11 sends this character. The process control system displays the data on the terminal but does not store it in memory.

5. E <ESC>: This is end of transmission. This character is sent by either computer immediately following the end of a file being transferred. In the Run Mode, this character is sent when the mode is exited. Upon receipt of this character from the process control system, the PDP-11 closes its file for input. When the PCS receives this character it returns to polling the port controllers.

PROGRAM LISTING

Copies of the process control system program listings may be obtained from Professor William E. Moritz, Department of Electrical Engineering, University of Washington.

Appendix B contains the process control system memory map, showing the memory allocations to program segments and data structures.

Chapter 5

CONCLUSIONS

SYSTEM TESTING

The process control system was tested by simulation. The Configuration Mode and Modify Mode were tested by entering and revising typical configuration files similar to those expected to be used with the system. In addition, arbitrary entries for files using the full 32 blocks available were successfully entered and changed. Incorrect entries for block types, inputs, gains, etc. were attempted and were succesfully caught by the system.

The Terminal Mode was tested by simulating the PDP-11 computer with a second terminal. Characters and protocol characters were correctly passed in each direction. The processer correctly identified the actions required by the protocol characters and successfully transferred files.

The Run Mode was tested by creating a simple configuration file to perform a single block operaton. The block type and parameters were varied and the control signal sent out was compared to a predicted value. A variable voltage power supply was used to provide the simulated sensor input. For the PID block type, a constant error was given a block operating as a PI controller. Because of the integral control action the block output for a constant input

is a linear ramp. This condition was obtained. A second block was added, defined as a PID block, operating to give PD type control. The ramp output of the first was used as input to the second. The derivative action of the second block provides a constant output for the ramp input. This condition was also obtained. These tests were all run in real time at a sample rate of 1 second. The system was also run at the limits of 0.1 and 10.0 seconds, and demonstrated that it did perform with these sampling periods.

ADVANTAGES OF THE SYSTEM

The system testing pointed out that a block diagram of a control strategy for a given physical process could be quickly and easily entered into the process control system configuration file. Most changes to the file were also easy to make. The editing feature available during the Run Mode is also a significant advantage. This permits "manual" control of all constant values such as set points, gains and PID reset and rate while the process is being controlled.

The system is "user-friendly." The menus of choices offered for the selection of operating modes and for the definition of elements of the control strategy relieve the user from continually referring to a manual. The questions, answers and error messages are all in English. To

help maintain the friendly atmosphere the rubout key is enabled so the user can correct typographical errors before the system sees the input. If an incorrect entry should be made, the system asks politely that the information be reentered.

The process control system is very versatile. It offers the basic PID control algorithm with extensive variation permitted by the use of the mathematical functions also offered. As a result, very complex control strategies can be realized through interconnection of up to 32 blocks in any combination of the 7 possible block types. For example, cascade control, where the process is controlled by several controllers in series, with the output of each becoming the setpoint of the next, can easily be provided by the system, up to a maximum of 16 controllers. In addition, with 4 separate output channels, up to 4 independent processes may be controlled simultaneously, although the same sampling rate must be used for all of them.

A feature common to many larger systems, but not generally available to small systems such as this one is data storage. The process control system can sample and store up to six variables in the control strategy each sampling period. This feature is particularly important in the use of the system in the Chemical Engineering Department to analyze the transient response of a control strategy. The

action of the controlled variable as a function of time needs to be stored to permit plotting and numerical analysis.

DISADVANTAGES OF THE PROCESS CONTROL SYSTEM

The system has limited file handling capabilities. It is therefore forced to deal with fixed length records and a fixed length configuration file. While the fixed length indexed sequential file type permits easy modification of the software to increase or decrease the number of blocks or system channels, such a change cannot be accomplished on-line. While this has little effect on the operation of the system, it results in one awkward situation for the user. If, in modifying a configuration, the user decides a block is no longer needed, he "removes" the block by no longer using the output of the block in the control strategy. The block remains in the file, but is not used. The system has no means of file compression to actually remove the block and move blocks forward in the file, filling the hole and making the necessary revisions of block numbers throughout the configuration file.

Symbols are very limited within the system. The system has defined the 16 input channels as S00 through S15, the output channels as C00 through C03, the 31 blocks as B00 through B31 and the 6 data points as D00 through D05.

While they serve to identify the various elements and are not foreign to engineers with some exposure to computer languages such as Fortran and Basic, they do not convey as much information about the process being controlled as user defined sysmbols.

Not all possible control algorithms are obtainable with the system. In particular, the dead-time control algorithm, which delays the control output signal an amount of time from when the sensor input was read, has not been implemented here, although the system is certainly capable of providing it.

POTENTIAL IMPROVEMENTS

The addition of more block types would further enhance the versatility of the system. The most return would be realized by addition of a block providing a dead-time or lead-lag algorithm. A dead-time control algorithm would require an extensive amount of memory to implement a circular buffer to store the sequence of inputs to the block. Consideration should be given to either limiting the number of possible dead-time blocks the system could provide or else defining the time delay in terms of sample periods rather than total time. To illustrate the need for this, at a 0.1 second sampling period, a 2 minute delay would require a 4K-byte circular buffer, 20% of the total avail-

able RAM. The same delay at a 10.0 second sampling period would require only 48 bytes of storage. By limiting the block to providing a maximum delay of 100 sample periods, a buffer of 400 bytes could be used and as many as 10 dead-time blocks could be used in a given configuration. In this manner, long delays would be available, but would require use of long sampling periods. Likewise, very short delays would require use of short sampling periods.

The addition of a mass storage device, such as a floppy disk system, would simplify creating and storing several control strategies for evaluation. Also, limitations on memory size, as mentioned above, could be removed by using the disk as virtual memory. Review of data collected could be done immedidately upon completion of an experiment, using the system terminal for data display.

One final, and extensive improvement, could be made to the process control system by the addition of a real time operating system. The process control system could then run as a real time application program under the operating system and use the operating system's file handling and other utility routines to remove the current limitations in file handling.

LIST OF REFERENCES

1. Shinskey, F. G., "Process-Control Systems." McGraw-Hill, 1967.

2. Luyben, W. L., "Process Modeling, Simulation and Control for Chemical Engineers." McGraw-Hill, 1973.

3. "Linear Databook." National Semiconductor Company, 1980.

4. Lapidus, Gerald, "Minicomputers--What All the Noise Is About." Control Engineering, September, 1968, pp. 73-80.

5. Kompass, E. J., "A Survey of On-Line Control Computer Systems." Control Engineering, January, 1972, pp. 52-56.

6. Andreiev, N., "Programmable Logic Controllers--An Update." Control Engineering, September, 1972, pp. 45-47.

7. Kompass, E. J. and Morris, H. M., "Comparing the Relative Complexities of Programming Process Controllers." Control Engineering, July, 1981, pp. 75-78.

8. "Systems Data Catalog." Intel Corporation, 1981.

9. "The Complete Motorola Microcomputer Data Library." Motorola, Inc., 1978.

10. "iSBC 80/24 Single Board Computer Hardware Reference Manual." Intel Corporation, 1981.

11. "iSBC 104/108/116 Combination Memory and I/O Expansion Board Hardware Reference Manual." Intel Corporation, 1979.

12. "Am9511A Arithmetic Processor" Product Note, Advanced Micro Devices, 1979.

13. "iSBC 711 Analog Input Board Hardware Reference Manual." Intel Corporation, 1977.

14. "iSBC 724 Analog Output Board Hardware Reference Manual." Intel Corporation, 1980.

15. "iSBC 660 System Chassis Hardware Reference Manual." Intel Corporation, 1980.

Appendix A

MICROCOMPUTER BASED PROCESS CONTROL SYSTEM

USER'S MANUAL

By Frank J. Nelson

August 18, 1982

INTRODUCTION

The process control system provides a means for easily defining, modifying and executing real time process control for a physical process. The system consists of a microcomputer, a video terminal, a 16 channel A/D converter, a 4 channel D/A converter, a panel for connecting the system to the process and the software which configures the system to the user's requirements. The system block diagram is shown in Figure A.1.

A resident real time control program relieves the user of having to do any programming to use the system. Instead, the user describes the block diagram of his desired control strategy in terms of "functional blocks," each of which provides a single mathematical or control operation. The system asks a series of questions to learn the types, inputs and other parameters which completely describe each block needed in the strategy and its relationship to other blocks and the process. An example of a single loop feedback control strategy is shown in Figure A.2.

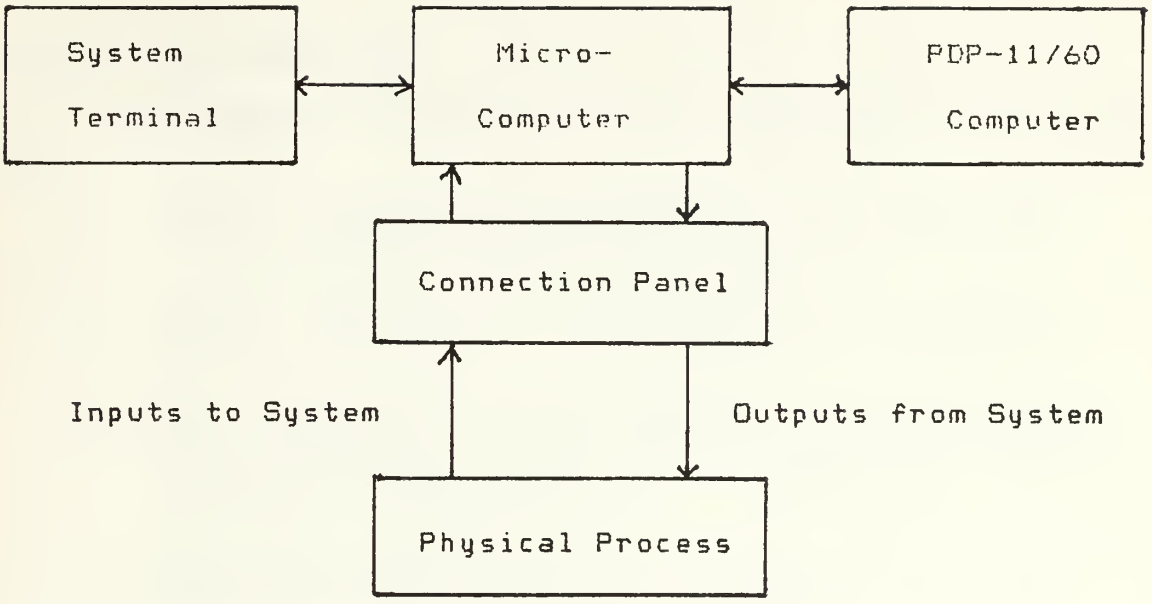
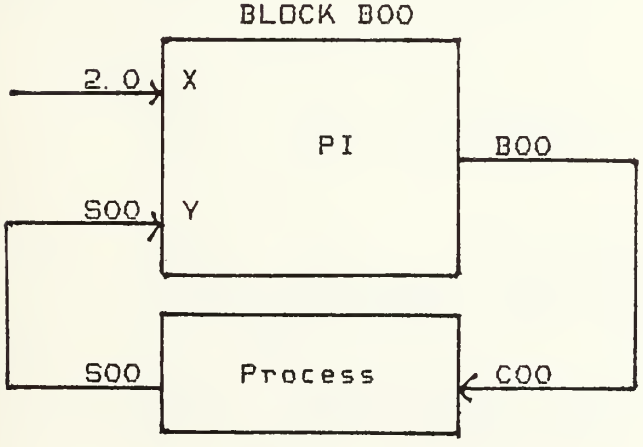


Figure A.1. System Block Diagram

Control Desired: Proportional-integral, with unity gain, a reset of 1/120 seconds and a setpoint of 2.0.



B00 Parameters

Type:	PID
X:	2.0
Y:	S00
Gain:	1
Constant:	NO
Reset:	0.0083
Rate:	0

Figure A.2. Single Loop Feedback Example

SYSTEM OPERATING MODE

The 4 modes of operation of the system and their uses are:

1. FILE: Create configuration file which describes desired control strategy.

2. MODIFY: Edit the configuration file currently in the system.

3. RUN: Execute real time control over the process connected to the system according to the strategy described in the configuration file currently in the system.

4. TERMINAL: System terminal becomes remote terminal of PDP-11/60 computer system.

FUNCTIONAL BLOCKS

A control strategy may use from 1 to a maximum of 32 blocks. The blocks are numbered by the system from B00 to B31. During real time control of a process, the system performs the block operations in block number order. The user is advised to define blocks such that a block's output will not be used as an input to a lower numbered block.

The constants used with a block, such as a set point input, are decimal numbers. The numbers may be signed, may be up to a maximum of 5 digits and may have the decimal place anywhere among the digits.

The parameters that define a block are:

1. TYPE: Type of function to perform. Each type has a three letter name.

2. INPUTS: X, Y. All type except SQT and SCL use both inputs. An input may be any system input channel, any block output or a constant. (X and Y may not both be constants.)

3. GAIN: The results of the function performed on the inputs to the block is multiplied by the block gain. The gain is a decimal constant.

4. CONSTANT OUTPUT: This is an option for a block. If selected, the output of the block is

held equal to a constant value, regardless of the inputs to the block. It is used to set initial conditions for the RUN mode, where the option can be removed when desired.

5. OUTPUT CONSTANT: This is the constant value used by a block selected for constant output. It is required only when the option is chosen.

6. RESET: This is R (or 1/Ti) for a PID block. It only appears for blocks defined as PID control blocks. It is a positive constant between 0 and 99999 (1/seconds).

7. RATE: This is D (or Td) for a PID block. It also only appears for blocks defined as PID control blocks. It is a positive constant between 0 and 99999 (seconds).

The block types available and their functions are:

1. ADD: Adds X and Y inputs.

2. SUB: Subtracts Y input from X input.

3. MUL: Multiplies X input by Y input.

4. DIV: Divides X input by Y input.

5. SQT: Takes square root of X input.

6. SCL: Scales X input by block gain.

7. PID: Provides P/PI/PID algorithm.

NOTE: Algorithm subtracts process variable, Y input, from set point, X input.

SYSTEM START-UP

The system is started by first tuning on power to the system terminal then to the microcomputer. After power has been applied, press the reset button on the microcomputer to reset the hardware and software. The system terminal will display the system menu. Any time the system menu is displayed, typing in a system command will initiate the associated mode of operation. The system menu, as seen when the system is ready, is shown in Figure A.3.

```
PCS V1.1

PLEASE ENTER SYSTEM COMMAND DESIRED, FOLLOWED BY <CR>

FILE    CREATE PROCESS CONTROL CONFIGURATION FILE

MOD     MODIFY CURRENTLY DEFINED CONFIGURATION FILE

PDP     ACCESS PDP-11/60 AS A REMOTE TERMINAL

RUN     RUN CONFIGURATION FILE AS A REAL TIME PROGRAM

EXIT    STOP FUNCTION AND RETURN TO HERE

*


"*" is system prompt for input from user.
"< >" identifies special key.
```

Figure A.3. System Menu

CREATING AND MODIFYING A CONFIGURATION FILE

Creating and modifying a configuration file are initiated by separate system commands, FILE and MOD, respectively, but are very similar in operation. They proceed through the same sections, offer the same questions and prompts and are interested in generally the same information. The two major differences between them are that in modifying a file, the old answer to a question is displayed in addition to the question, and in the modify mode, skipping past questions, blocks and sections is permitted to speed access to the material to be changed.

Creating and modifying a configuration file are handled according to the following sections:

1. File Name: Identifies file for user convenience. Optional.

2. Data File Name: Name of file being used to save data. This provides a record of the data file associated with this configuration for the user's convenience. Optional.

3. Sampling Period: The sampling period of the

real time process control action.

4. Block Bxx Definition: Defines the blocks used in the control strategy. Blocks are presented sequentially, beginning with B00.

5. Output Cxx Definition: Defines which points within the control strategy to connect to the process through the output control channels. Channels are presented sequentially, beginning with C00.

6. Data Point Dxx Definition: Defines points in the control strategy to display on the system terminal (and send to the PDP-11 if a file has been opened) during real time control of the process. Points are presented sequentially, starting with D00.

When modifying a file, the inputs used to skip past material are:

1. <CR> Skips to next question.

2. <LF> Skips to next block. (In sections without blocks, skips to next section.)

3. <ESC> Skips to next section.

An example of the dialog between the user and system in defining a block during file creation) is shown in Figure A.4.

```
FUNCTIONAL BLOCK DEFINITION,     MAXIMUM 32 BLOCKS
WHEN NO MORE BLOCKS DESIRED, ENTER <ESC> FOR TYPE
BLOCK NUMBER = B00
TYPE?  ADD, SUB, MUL, DIV, SQT, SCL OR PID <CR>
*PID <CR>
 --------
X INPUT?  S00-S15, B00-B31 OR MAX 5 DIGIT CONSTANT <CR>
*2.0 <CR>
 --------
Y INPUT?  S00-S15, B00-B31 OR MAX 5 DIGIT CONSTANT <CR>
*S00 <CR>
 --------
GAIN?  MAX 5 DIGIT CONSTANT <CR>
*1 <CR>
 ------
CONSTANT OUTPUT?  YES OR NO <CR>
*NO <CR>
 -------
PID RESET?  0 TO 99999 (1/SECS) <CR>
*0.0083 <CR>
 -----------
PID RATE?  0 TO 99999 (SECS) <CR>
*0 <CR>
 ------
```

User input is shown underlined.
"*" is system prompt.
"< >" identifies special key.

Figure A.4. Example of Block Definition

RUNNING A CONFIGURATION FILE

Executing real time control of a process is accomplished by selecting the system command RUN. The system will then ask for the command "GO," signifying the process is connected to the system and is ready. After entering "GO" the user may remove initial conditions, or make any other changes to the constants in the control strategy by entering the block number of the block to be changed. All constant output options can be removed simultaneously by entering "ALL" instead of a block number. An example of the terminal display seen while the system is executing a process control strategy is shown in Figure A.5.

```
RUN PROCESS CONTROL CONFIGURATION
TIME 02:37


D00 =  3.3234
D01 = -.00005
D02 =  1.0000


ENTER HALT <CR> TO STOP
ENTER BLOCK NUMBER <CR> TO CHANGE BLOCK
*


"*" is system prompt for input.
"< >" identifies special key.
```

Figure A.5. Example of Run Mode Display

The process control action can be stopped at any time by entering "HALT." After a halt, the process may be restarted (from the point where halted) or the user may exit the mode with the "EXIT" command.

REMOTE TERMINAL

The user may gain direct access to the PDP-11 computer by entering the system command PDP. In addition to being able to fully use the PDP-11 operating system, the user may also exchange configuration and data files between the two systems. This is accomplished by running the PDP-11 program "$PCS" from the terminal. The program enables configuration files to be stored on the PDP-11 system and later down-loaded back to the process control system. It also enables data files to be set up to receive real time data from the process control system. Further details concerning the $PCS program can be obtained from the PDP-11/60 System Manager. This mode is exited by entering the escape character <ESC>.

Appendix B

MEMORY MAP

Table B.1 provides a map of the process control system memory.

Start Address	Length	Use	
0000H	3H	Absolute:	Start from zero.
003CH	3H	Absolute:	Interrupt Vector.
0040H	14EFH	Code:	Includes Modules:
0040H	E6H		PCSEXC.OBJ
0226H	2A5H		PCSCFG.OBJ
04CBH	B9H		PCSRUN.OBJ
0584H	137H		PCSPDP.OBJ
06BBH	16BH		LIBIOR.OBJ
0826H	22AH		LIBREC.OBJ
0A50H	77DH		LIBFIL.OBJ
11CDH	1F2H		LIBFLT.OBJ
13BFH	170H		LIBRUN.OBJ
152FH	CH	Stack	
3000H	1100H	Memory:	Open
4100H	CD8H	Data:	Includes Buffers:
413FH	8AH		LINE
41A9H	20H		ADCBUF
41C9H	8H		DACBUF
41D1H	354H		Run Tables
4525H	8B3H		CFG File
4DD8H	3328H	Memory:	Open

Table B.1. Process Control System, Memory Map

Thesis 199473
N3614 Nelson
c.1 The design of a
microcomputer based
generalized process
control system.